普通高等教育“十一五”规划教材

神经网络实用教程

张良均 曹晶 蒋世忠 编

主审 胡学钢



机械工业出版社

内容简介：

本书基于 MATLAB 6.5/7 提供的神经网络工具箱，介绍了神经网络常用算法、优化算法及其混合编程实现。全书共分为 6 章，分别结合实例介绍了人工神经网络概述，实用神经网络模型与学习算法，神经网络算法优化，nnToolKit 神经网络工具包，MATLAB 混合编程技术，混合编程案例。附录中介绍了 2NDN 神经网络建模仿真平台。全书图文并茂，由浅入深，脉络清晰，融教学与实例于一体，通过大量的神经网络应用实例介绍了神经网络的常用算法及混合编程实现方法，并配有习题。全书可读性和操作性较强。

本书可作为高校自动化、计算机、材料化工、机械工程、数学、电子工程、信息与信息处理等专业的教材和相关专业工程技术人员的参考书，读者可到中国 AI 创业研发俱乐部（http://www.2nsoft.cn）网站上下载本书配套源程序和电子课件，同时本网站的“源码中心”提供了大量的神经网络源代码，欢迎大家下载交流学习。

前言

自20世纪80年代末以来，神经网络这个涉及多种学科的新的高科技领域，吸引了众多的神经生理学家、心理学家、数理学家、计算机与信息科学家及工程师和企业家等进行研究和应用。大量的有关神经网络机理、模型、算法特性分析，以及在各领域应用的学术论文像雨后春笋般在报刊杂志上和许多国际学术会议中涌现。神经网络日益成为当代高科技领域中方兴未艾的竞争热点。

MATLAB是一款强大的工程计算和仿真软件，其中的神经网络工具箱提供了大量可直接调用的函数和命令，基本上囊括了目前应用比较成熟的神经网络设计方法。用 MATLAB 来编写各种网络仿真和训练程序，可以使用户从繁琐的编程中解脱出来，大大提高工作效率和解题质量。因此，如何应用神经网络工具箱函数来解决工程实践中的问题已成为燃眉之急。作者根据多年来从事神经网络应用开发以及使用 MATLAB 的经验编写了这本书，书中除了介绍神经网络的实用算法外，还详细介绍了在高级编程语言中神经网络的混合编程，并配有丰富的实例。

全书内容主要包括：人工神经网络概述，实用神经网络模型与学习算法，神经网络算法优化，nnToolKit 神经网络工具包，MATLAB 混合编程技术，混合编程案例，2NDN 神经网络建模仿真平台。第1章人工神经网络概述，介绍了神经网络的基础知识、特点以及应用，使得初学者对神经网络有个基本的了解；第2章实用神经网络模型与学习算法，详细地介绍了常用神经网络模型与学习算法，并给出了完整的示例；第3章神经网络算法优化，重点介绍了BP网络学习算法的改进算法，模糊神经网络，基于遗传算法的神经网络训练方法(GA-BP 法)，小波神经网络；第 4 章 nnToolKit 神经网络工具包，重点介绍 nnToolKit 神经网络工具包。nnToolKit 是一个基于 MATLAB 神经网络工具箱编写的常用神经网络函数的集合，其M文件可单独运行，可编译成MEX、C、CPP，或直接封装成DLL库，在脱离MATLAB环境的情况下，直接供高级语言调用，并给出工程实践中的4个完整实例：基于LM算法进行房地产开发风险预测，模糊神经网络预测地基沉降量，基于遗传神经

网络的图像分割，小波神经网络在 1-D 插值上的应用。第 5 章 MATLAB 混合编程技术，详细介绍了 MATLAB COM Builder、Excel Builder 等 Matlab 混合编程技术，同时介绍了一个案例——电力行业的漏窃电预测。第 6 章神经网络混合编程案例，完整地介绍了一个案例——个人资信评级系统。附录 2NDN 神经网络建模仿真工具介绍了目前国内较受欢迎的，由作者开发的 2NDN 神经网络建模仿真平台，这个平台能帮助使用者大大提高开发人工神经网络的效率。全书通过翔实的神经网络混合编程实例为读者讲述了神经网络的常用算法及混合编程实现方法，全书图文并茂，深入浅出，脉络清晰，可读性强。相信广大读者通过认真学习本书，能快速学会神经网络技术及其混合编程实现方法。

本书由中国 AI 创业研发俱乐部策划并组织编写，张良均、曹晶、蒋世忠负责全书的编写与审校工作，第 1 章由曹晶、蒋世忠编写，第 2 章由蒋世忠、曹晶编写，第 3 章由蒋世忠、张良均编写，第 4、5 章由曹晶编写，第 6 章及附录由曹晶、张良均编写，全书由张良均统稿。中国 AI 创业研发俱乐部会员田大新、程起才、李刚、谭显胜、冰露、田慧欣、马琼雄、李增祥、李超、杨富森、刘宇、潘永刚等为图书的组织及书稿的材料整理提供了帮助与意见，另外，还有不少俱乐部会员在本书的编写过程中也付出了自己的劳动。在本书的编写过程中，合肥工业大学计算机学院胡学钢教授，给出了具体的指导意见，广州万友人工智能软件有限公司提供了大量的神经网络源程序及详尽案例，在此一并表示衷心的感谢。

本书可作为高等学校理工类各专业高年级本科生和研究生的神经网络课程的教材，也可作为各领域工程技术人员的参考用书，还可作为其他科技工作者应用神经网络的参考资料。

由于时间仓促加之作者本身水平有限，书中错误之处在所难免。在此，敬请各领域专家和广大读者批评指正。

联系方式如下：咨询电话：（020）33118470　　电子邮件：service@2nsoft.cn

俱乐部网址：http://www.2nsoft.cn　http://www.2nsoft.cn/bbs

编者

目录

第 1 章　人工神经网络概述

人工神经网络（Artificial Neural Networks，ANNs），也简称为神经网络（NNs），是模拟生物神经网络进行信息处理的一种数学模型。它以对大脑的生理研究成果为基础，其目的在于模拟大脑的某些机理与机制，实现一些特定的功能。目前，人工神经网络已应用于很多领域。本章主要对人工神经网络的基本理论做一个全面简要的介绍。

1.1　神经网络的基本概念

1.1.1 生物神经元的结构与功能特点

人工神经网络从生物神经网络发展而来，一个神经元就是一个神经细胞，在人类大脑皮层中大约有 100 亿个神经元，60 万亿个神经突触以及它们的连接体。

神经元是基本的信息处理单元。生物神经元主要由细胞体、树突、轴突和突触组成。

(1)细胞体　细胞体是神经元的主体。由细胞核、细胞质和细胞膜 3 部分构成。它是神经元活动的能量供应地，也是进行新陈代谢等各种生化过程的场所。

(2)树突　从细胞体向外延伸出许多突起的神经纤维，这些突起称为树突。神经元靠树突接受来自其他神经元的输入信号，相当于细胞体的输入端。

(3)轴突　由细胞体伸出的最长的一条突起称为轴突，轴突比树突长而细，用来传出细胞体产生的输出电化学信号，相当于细胞体的输出端。

(4)突触　神经元之间通过一个神经元的轴突末梢和其他神经元的细胞体或树突进行通信连接、这种连接相当于神经元之间的输入输出接口，称为突触。

现代生理学研究已经证明：人类大脑的活动，不是一个生物神经元所能完成的，也不是多个生物神经元功能的简单叠加，而是多单元的非线性的动态处理系统。

1.1.2 人工神经元模型

人工神经元是人工神经网络操作的基本信息处理单位。人工神经元的模型如图 1-1 所示，它是人工神经网络的设计基础。

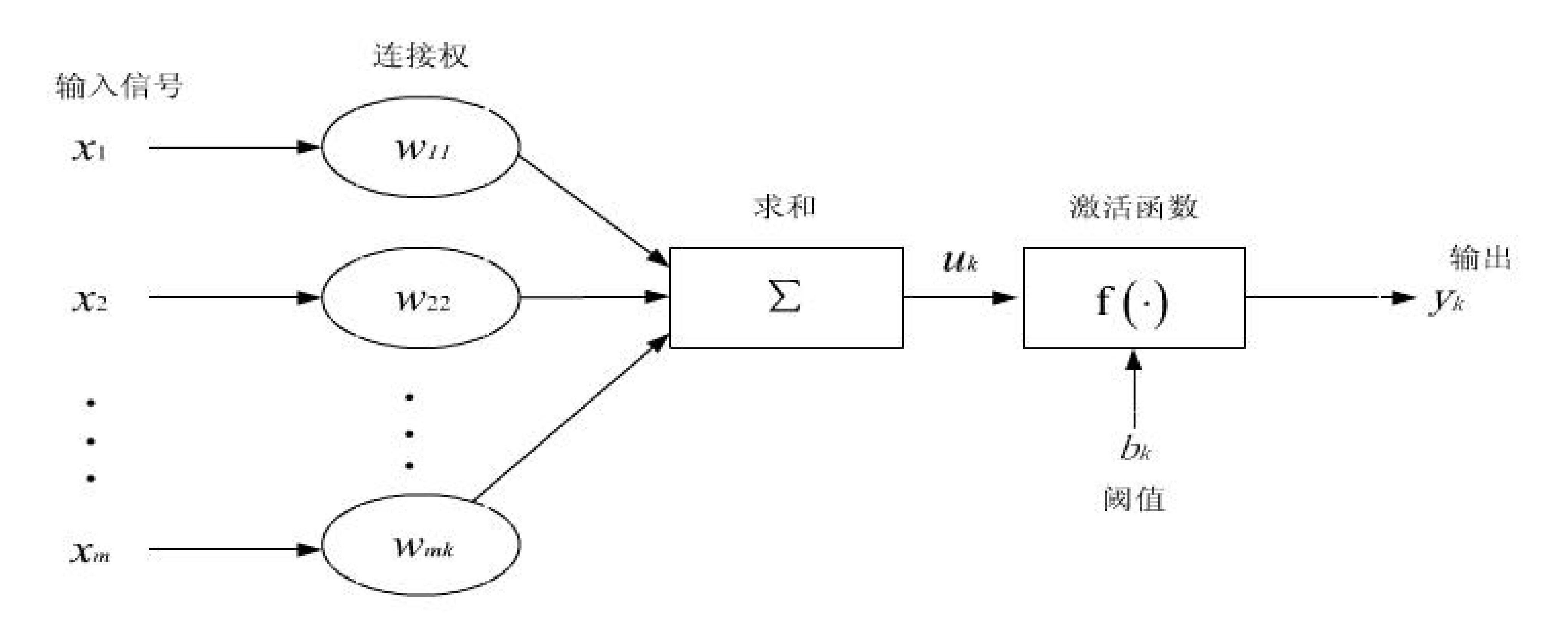


图 1-1 人工神经元模型

人工神经元模型可以看成是由 3 种基本元素组成：

(1)一组连接 连接强度由各连接上的权值表示，权值可以取正值也可以取负值，权值为正表示激活，权值为负表示抑制。

(2)一个加法器 用于求输入信号对神经元的相应突触加权的和。

(3)一个激活函数 用来限制神经元输出振幅。激活函数也称为压制函数，因为它将输入信号压制(限制)到允许范围之内的一定值。通常，一个神经元输出的正常幅度范围可写成单位闭区间[0，1]，或者另一种区间[-1，+1]。

另外，可以给一个神经元模型加一个外部偏置，记为 b_k。偏置的作用是根据其为正或为负，相应地增加或降低激活函数的网络输入。一个人工神经元 k 可以用以下公式表示：

$$u_k = \sum_{i=1}^{m} w_{ik} x_i$$

$$y_k = \mathrm{f}(u_k + b_k)$$

式中 $x_i(i = 1,\cdots$ 输入信号；

$w_{ik}(i=1,\cdots$　　神经元k的突触权值（对于激发状态，w_{ik}取正值；对于抑制状态，w_{ik}取负值，m为输入信号数目）；

u_k　输入信号线性组合器的输出；

b_k　神经元单元的偏置（阈值）；

f()　激活函数；

y_k　神经元输出信号。

激活函数主要有以下 3 种形式，其中，$v=u_k+b_k$

(1)域值函数 即阶梯函数，当函数的自变量小于 0 时，函数的输出为 0；当函数的自变量大于或等于 0 时，函数的输出为 1。用该函数可以把输入分成两类：

$$\mathrm{f}(v)=\begin{cases}1 & v\geq 0\\ 0 & v<0\end{cases}$$

(2)分段线性函数 该函数在（-1，+1）线性区内的放大系数是一致的，这种形式的激活函数可以看作是非线性放大器的近似，如图 1-2a 所示。

$$\mathrm{f}(v)=\begin{cases}1, & v\geq 1\\ v, & -1<v<1\\ -1, & v\leq -1\end{cases}$$

(3)非线性转移函数 该函数为实数域 R 到[0，1]闭集的非连续函数，代表了状态连续型神经元模型。最常用的非线性转移函数是单极性 Sigmoid 函数曲线，简称 S 型函数，其特点是函数本身及其导数都是连续的，能够体现数学计算上的优越性，因而在处理上十分方便。单极性 S 型函数定义如下：

$$\mathrm{f}(v)=\frac{1}{1+\mathrm{e}^{-v}}$$

有时也采用双极性 S 型函数（即双曲正切）等形式：

$$\mathrm{f}(v)=\frac{2}{1+\mathrm{e}^{-v}}-1=\frac{1-\mathrm{e}^{-v}}{1+\mathrm{e}^{-v}}$$

单极 S 型函数曲线特点如图 1-2b 所示。



图 1-2 激活函数

a) 分段线性函数　　b) 单极 S 型函数

1.1.3 神经网络的结构及工作方式

如果将大量功能简单的神经元通过一定的拓扑结构组织起来，构成群体并行式处理的计算结构，则这种结构就是人工神经网络。

将一个神经元的输出送到另一个神经元作为输入信号称之为连接，每个连接通路对应一个连接权系数，相同神经元经过不同的连接方式将得到具有不同特性的神经网络。

根据神经元的不同连接方式，可将神经网络分为两大类：分层网络和相互连接型网络。

1.分层网络

分层网络将一个神经网络模型中的所有神经元按照功能分成若干层。一般有输入层、隐含层(中间层)和输出层，各层顺次连接。

输入层接收外部输入模式，并由各输入单元传送给相连的隐含层各单元；隐含层是神经网络的内部处理单元层，神经网络所具有的模式变换能力，如模式分类、模式完善、特征抽取等，主要体现在隐含层单元的处理，根据模式变换功能的不同，隐含层可以有多层，也可以没有一层；输出层产生神经网络的输出模式。

分层网络可以细分为 3 种互联方式：

(1)单纯的前向网络(见图 1-3a) 输入模式由输入层进入网络，经过中间各层的顺序模式变换，由输出层产生一个输出模式，便完成一次网络状态的更新。

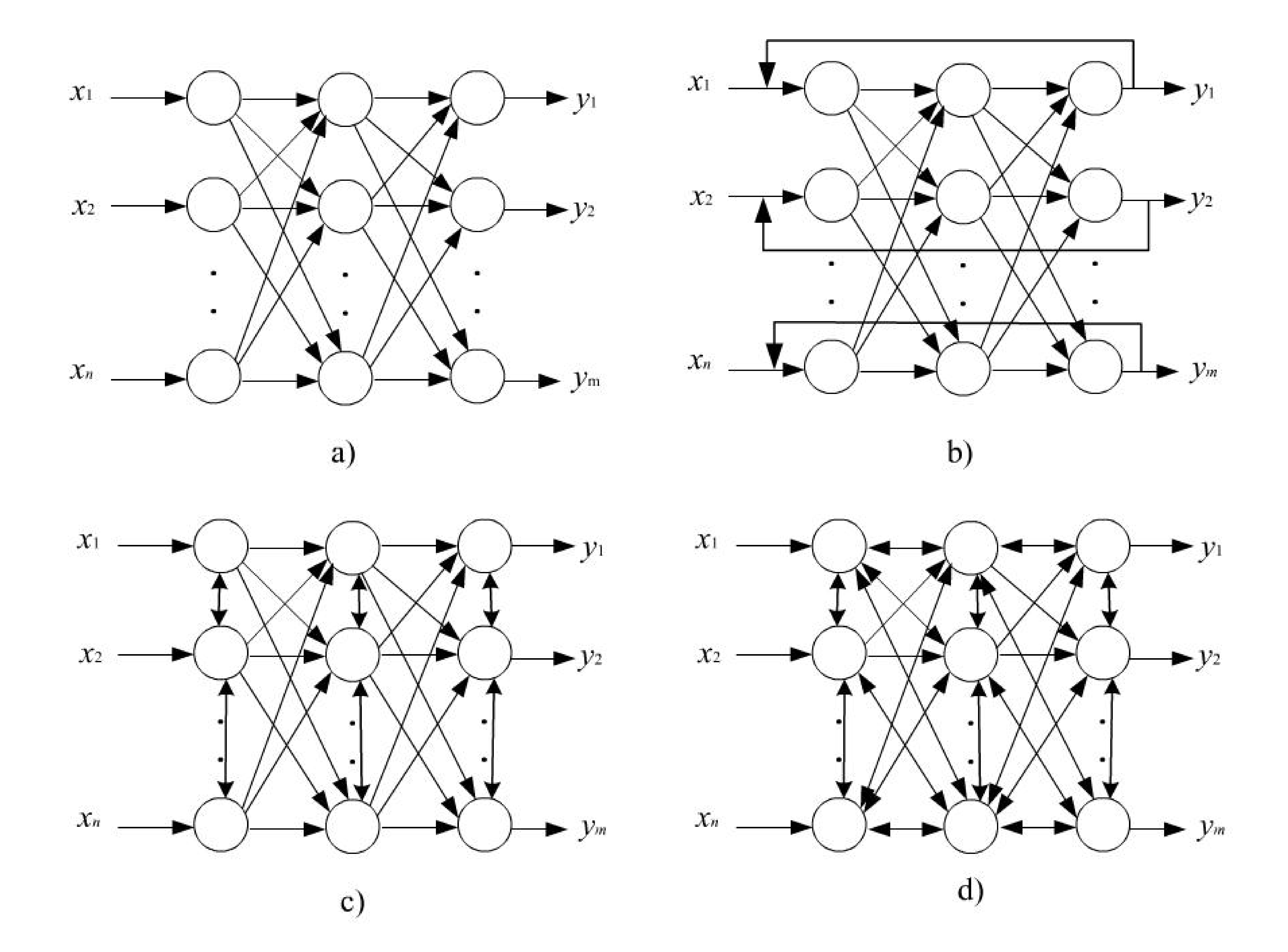


图 1-2 神经网络的连接方式

a）单纯的前向网络 b）具有反馈的前向网络 c）层内互联前向网络 d）互联网络

(2)具有反馈的前向网络(见图 1-3b) 反馈的结构形成封闭环路，具有反馈的单元也称为隐单元，其输出称为内部输出，而网络本身还是前馈型的。

(3)层内互联的前向网络(见图 1-3c) 同一层内单元的相互连接使它们彼此之间相互制约，如限制同一层内能同时激活的单元个数，而从外部看来还是前向网络。一些自组织竞争网络就采用这种拓扑结构。

2.相互连接型网络

如图 1-3d 所示，所谓相互连接是指网络中任意两个单元之间都是可达的，即存在连接路径。互连网络又分为局部互连和全互连。全互连网络中每个神经元的输出都与其他神经元相连，而局部互连网络中，有些神经元之间没有连接关系。

对于简单的前向网络，给定某一输入模式，网络能迅速产生一个相应的输出模式，并保持不变。但在相互连接的网络中，对于给定的某一输入模式，由某一网络参数出发，在一段时间内处于不断改变输出模式的动态变化中，网络最终可能产生某一稳定的输出模式，但也可能进入周期性振荡或混沌状态。

1.1.4 神经网络的学习

1.学习方式

神经网络的学习也称为训练，指的是神经网络在受到外部环境的刺激下调整神经网络的参数，使神经网络以一种新的方式对外部环境作出反应的一个过程。

能够从环境中学习和在学习中提高自身性能是神经网络最有意义的性质,神经网络经过反复学习来达到对环境的了解。

神经网络的学习方式可分为有导师学习、无导师学习和再励学习。

(1)有导师学习 亦称监督学习，它需组织一批正确的输入输出数据对。将输入数据加载到网络输入端后，把网络的实际输出与期望(理想)的输出相比较得到误差，然后根据误差的情况修改各连接权值，使网络能朝着正确响应的方向不断变化下去，直到实际的输出与期望输出之差在允许范围之内。

(2)无导师学习 亦称无监督学习，这时仅有一批输入数据。网络初始状态下，连接权值均设置为一小正数，通过反复加载这批输入数据，使网络不断受到刺激，当与曾经历的刺激相同的刺激到来时，响应连接权以某一系数增大，重复加入的同样刺激使相应的连接权增大到接近 1 的某值。这一自组织的方法，使网络具有某种“记忆”能力以至形成“条件反射”，当曾经学习过或相似的刺激加入后，输出端便按权值矩阵产生相应的输出。

(3)再励学习 亦称强化学习。这种学习介于上述两种情况之间，外部环境对系统输出结果只给出评价(奖和罚)而不是给出正确答案，学习系统通过强化那些受奖励的动作来改善自身性能。

2.学习算法

学习算法是指针对学习问题的明确规则，学习类型是由参数变化发生的形式决定的，不同的学习算法对神经元的权值调整的表达式是不同的。没有一种独特的学习算法适用于设计所有的神经网络。选择或设计学习算法时还需要考虑神经网络的结构及神经网络与外界环境相连接的形式。

(1)Hebb 学习规则 它是 D.O.Hebb 根据生物学中条件反射机理，于 1949 年提出的神经元连接强

度变化的规则，属于无导师学习。其内容为:如果两个神经元同时兴奋，则它们之间的突触连接加强。如果神经元i是神经元j的上层结点，用v_i, v_j表示神经元i和j的激活值(输出)，w_{ij}表示两个神经元之间的连接权，则 Hebb 学习规则可以表示为

$$\Delta w_{ij} = \eta v_i v_j$$

式中η表示学习速率。Hebb 学习规则是人工神经网络学习的基本规则，几乎所有神经网络的学习规则都可以看作 Hebb 学习规则的变形。

(2)δ 学习规则(误差校正学习算法) 误差校正学习算法的适用面比较广一些，它可用于非线性神经元的学习过程。它学习样本的数量也没有限制，甚至于它还容忍训练样本中的矛盾之处，这也是神经网络容错性能的表现方式之一，误差校正学习算法是根据神经网络的输出误差对神经元的连接强度进行修正，属于有导师学习。设$(\mathbf{X}^k, \mathbf{D}^k), k=1,2,\cdots$为输入输出样本数据对，其中，$\mathbf{X}^k = [x_1, x_2, \cdots, x_m]^T$，$\mathbf{D}^k = [d_1, d_2, \cdots, d_p]^T$。把$\mathbf{X}^k$作为网络的输入，在连接权的作用下，可得到网络的实际输出$\mathbf{Y}^k = [y_1, y_2, \cdots, y_p]^T$。设神经元$i$到$j$的连接权为$w_{ij}$，则权的调整量为

$$\Delta w_{ij} = \eta \delta_j v_i$$

$$e = \frac{1}{2}\sum_{o=1}^{q}(d_0(k) - y_0(k))^2$$

式中，η为学习速率；δ_j为误差函数对神经元j输入的偏导数；v_i为第i个神经元的输出。它是神经网络中非常重要的一类算法，前馈网络的 BP 算法即是δ 学习规则。

(3)随机学习算法 在上面谈到的误差学习算法通常采用梯度下降法，存在局部最小问题。随机学习算法通过引入不稳定因子来处理这种情况。如果把神经网络的当前状态看作一个小球，神经网络的误差函数看作是超平面，当小球达到局部最小值时，增加不稳定因子，即对小球加一个冲量，则小球会越过峰值点，而达到全局最小点，即神经网络最终收敛于全局最小点。一般而言，不稳定因子是从大到小逐渐变化的，只要其变化足够慢，学习时间足够长，总存在一种状态使得神经网络可从局部最小跳出，而无法从全局最小跳出，从而使神经网络收敛于全局最小点。比较著名的随机学习算法有模拟退化算法和遗传算法。

(4)竞争学习算法 有导师的学习算法不能充分反映出人脑神经系统的高级智能学习过程，人脑神经系统在学习过程中各个细胞始终存在竞争。竞争学习网络由一组性能基本相同，只是参数有所

不同的神经元构成。对于一个输入模式内各子模式的作用，每个神经元通过互相竞争来做出不同的反映，每个神经元的激活范围遵循某种特定的限制。

竞争学习的基本思想是:竞争获胜的神经元权值修正，获胜神经元的输入状态为 1 时，相应的权值增加，状态为 0 时权值减小。学习过程中，权值越来越接近于相应的输入状态。竞争学习属于无导师算法。Kohomen 提出的自组织特征映射网(Self-Organization Feature Map，SOM)及自适应共振网(Adaptive Resonace Theory，ART)均采用这种算法。

1.2 神经网络的特点及其应用

1.2.1 神经网络的特点

神经网络的基本属性反映了神经网络特点，主要表现在：

1．并行分布式处理 神经网络具有高度的并行结构和并行实现能力，具有高速寻找优化解的能力，能够发挥计算机的高速运算能力，可能很快找到优化解。

2．非线性处理 人脑的思维是非线性的，故神经网络模拟人的思维也应是非线性的。这一特性有助于处理非线性问题。

3．具有自学习功能 通过对过去的历史数据的学习，训练出一个具有归纳全部数据的特定的神经网络，自学习功能对于预测有特别重要的意义。

4．神经网络的硬件实现 要使人工神经网络更快、更有效地解决更大规模的问题，关键在于其超大规模集成电路（V LSI）硬件的实现，即把神经元和连接制作在一块芯片上（多为 CMOS）构成 ANN，神经网络的 VLSI 设计方法近年来发展很快，硬件实现已成为 ANN 的一个重要分支。

1.2.2 神经网络的应用领域

近些年来神经网络在众多领域得到了广泛的运用。在民用应用领域的应用，如语言识别、图像识别与理解、计算机视觉、智能机器人故障检测、实时语言翻译、企业管理、市场分析、决策优化、物资调运、自适应控制、专家系统、智能接口、神经生理学、心理学和认知科学研究等等；在军用应用领域的应用，如雷达、声纳的多目标识别与跟踪，战场管理和决策支持系统，军用机器人控制各种情况、信息的快速录取、分类与查询，导弹的智能引导，保密通信，航天器的姿态控制等。

练习题

1、什么是人工神经网络，研究人工神经网络的目的是什么？

2、人工神经网络是从哪些方面去模拟人的智能的？

3、如何理解人工神经网络的学习能力？有哪几种学习方法？

4、偏置的作用是什么？激活函数的作用是什么？激活函数有哪些主要形式？

第 2 章 实用神经网络模型与学习算法

人工神经网络具有大规模并行分布式结构、自主学习能力以及由此而来的泛化能力，因此可以利用人工神经网络来解决许多用传统方法无法解决的复杂问题。由于实际情况千差万别，目前还不存在适用于各种环境和情况的通用人工神经网络模型，为了解决实际问题，构建特定情况下的人工神经网络模型，就必须掌握一些常用的人工神经网络模型和学习算法，了解它们的特点和适用范围，在此基础上，将常用的人工神经网络做相应的修改并应用到所遇到的特定环境中，解决自己的实际问题。

本章将详细介绍几种常用神经网络的算法，着重讲解这些神经网络的特点、模型及其算法。MATLAB 是一款非常优秀的仿真工具，它的神经网络工具箱提供了许多进行神经网络设计和分析的工具函数，给使用人工神经网络来解决实际问题的用户带来了极大的方便，即使对计算机编程语言不熟悉，也可以直接应用功能丰富的函数来解决自己的实际问题。因此，本章在介绍常用神经网络的算法前，首先介绍 MATLAB 的使用方法，在介绍每一个常用的人工神经网络模型和学习算法后，给出了与之相对应的 MATLAB 神经网络仿真工具箱函数，并用对应的工具箱函数实现一个具体实例，读者可以参考这些实例，根据自己的需要去调用这些工具箱中的函数来创建自己的神经网络，也可以利用第 4 章介绍的 nnToolKit 神经网络工具包和第 5 章的 MATLAB 混和编程技术编写自己的函数，设计并实现适合自己领域内的特定问题的神经网络方案。

2.1 MATLAB 快速入门

熟悉 MATLAB 软件的基本功能与操作方式，对于正确快速地利用 MATLAT 中的神经网络工具箱函数来创建自己的神经网络将起到事半功倍的作用。下面将介绍 MATLAB 软件的主要操作界面的使用方法、基本运算与绘图函数。对于没有介绍的函数的功能、调用格式及详细使用方法，读者可以通过相关的 MATLAB 书籍或 MATLAB 的帮助文档得到。

2.1.1 MATLAB 界面组成

安装好 MATLAB 后，启动 MATLAB，则可以看到如图 2-1-1 所示的界面，界面主要由命令行窗口、命令历史窗口、工作空间浏览器窗口、当前目录窗口和编辑调试窗口所组成。

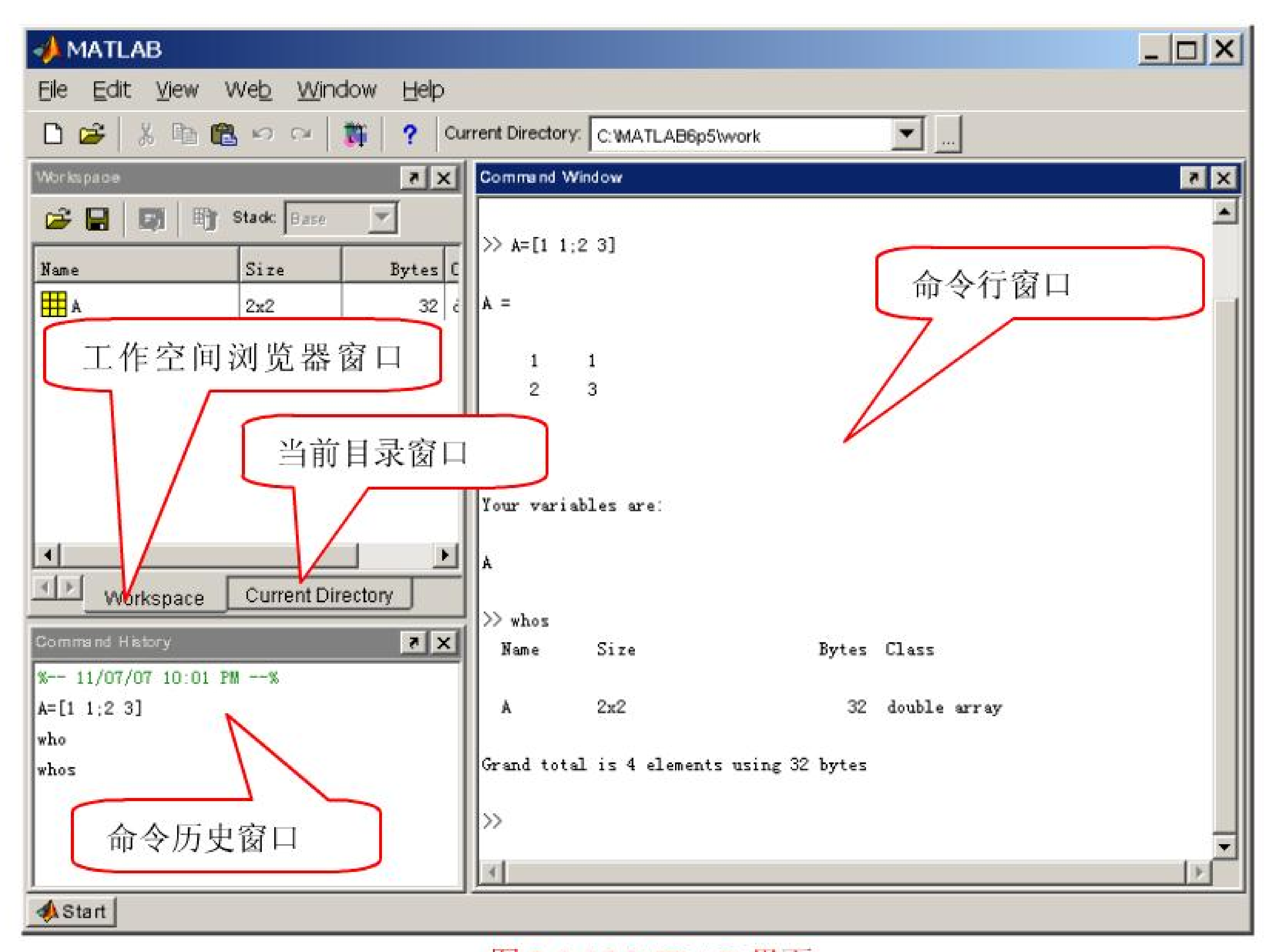


图 2-1-1 MATLAB 界面

1、 命令行窗口 是 MATLAB 的输入窗口，可以在此输入 MATLAB 的命令，该命令可以是一句 MATLAB 语句，也可以是一段利用 MATLAB 编程功能实现的代码。

2、 命令历史窗口 该窗口将保存在命令行窗口中输入的所有命令，如果需要重新运行以前输入的命令，只需在命令历史窗口中选中该命令，然后双击鼠标即可。若需重新运行多条命令，只需要按住键盘上的 Shift 键，然后选中需要重新执行的命令，双击选中的命令就行了。

3、 工作空间浏览器窗口 本窗口存储并显示当前命令行窗口中所有的变量，这些变量保存在计算机的内存中，在 MATLAB 进程结束以前，一直是活动的。在命令行窗口中输入 who 和 whos，可以查看当前内存中的所有变量，包括变量的名称、大小和类型。在命令行窗口中输入 clear，可以清除当前内存的所有变量。

4、 当前目录窗口 该窗口指示当前的工作目录的位置，如果不改动的话，默认的当前目录为安装 MATLAB 路径下的 work 文件夹。

5、 编辑调试窗口 在 MATLAB 主窗口中，点击“Start”→“Desktop Tools”→“Editor”即可打开编辑调试窗口，如图 2-1-2 所示。

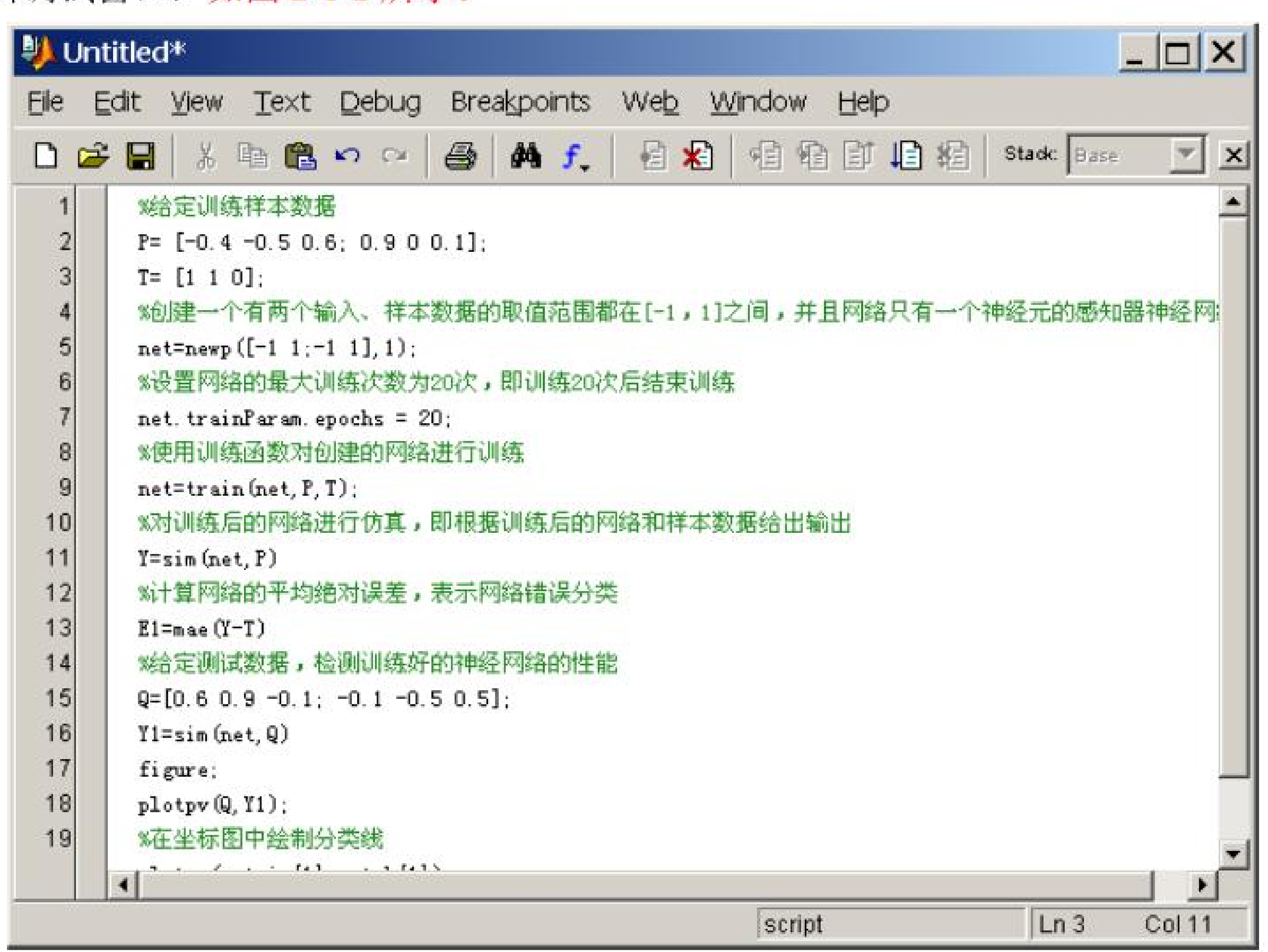


图 2-1-2 编辑调试窗口

编辑调试窗口用于 MATLAB 程序的编写和执行，当程序编写结束后，保存为一个 M 文件，此

时，可以从 Debug 菜单中选择 Run 命令，也可以直接按 F5 来执行编写好的程序。

2.1.2 MATLAB 基本运算

MATLAB 包括丰富的运算与函数，因篇幅问题，不可能在此一一列举。此节作为 MATLAB 的快速入门，仅介绍一些与后续章节联系比较密切的运算与函数。在讲述各种类型的神经网络时，遇到此章没有介绍的 MATLAB 函数时，也会作简要的介绍。

1、基本运算 在 MATLAB 下进行基本数学运算，只需将运算式直接打入提示号“>>”之后，并按回车键即可。例如：

```
>> (5*2+1.3-0.8)*10/25
ans =4.2000
```

MATLAB 会将运算结果直接存入一变量 ans，ans 代表 MATLAB 运算的答案并在屏幕上显示其数值。也可将上述运算式的结果赋给另一个变量 x：

```
x = (5*2+1.3-0.8)*10^2/25
x =
      42
```

此时 MATLAB 会直接显示 x 的值。在 MATLAB 中所使用变量，其命名的规则就遵循以下规则：

1）第一个字母必须是英文字母；

2）字母间不可留空格；

3）最多只能有 19 个字母，MATLAB 会忽略多余字母。

由上例可知，MATLAB 认识所有用到的 “+”、 “-”、 “*”、 “/” 的数学运算符号，以及幂运算符号“^”。MATLAB 将所有变量均保存成 double 的数据类型，所以不需经过变量声明。若不想让 MATLAB 每次都显示运算结果，只需在运算式最后加上 “；”即可，如下例：

```
y = sin(10)*exp(-0.3*4^2);
```

若要显示变量 y 的值，直接键入 y 即可：

```
>>y
y =
    -0.0045
```

在上例中，sin 是正弦函数，exp 是指数函数，这些都是 MATLAB 常用到的数学函数。

MATLAB 可同时执行数个命令，只要以逗号或分号将命令隔开：

```
x = sin(pi/3); y = x^2; z = y*10,
z =
    7.5000
```

若一个数学运算式太长，可用三个句点将其延伸到下一行：

```
z = 10*sin(pi/3)* ...
sin(pi/3);
```

2、MATLAB 的查询命令 若对 MATLAB 函数用法有疑问，可随时使用 help 来查询该函数的用法。例如已知 inv 是用来计算逆矩阵，键入 help inv 即可得知有关 inv 命令的用法。键入 help help 则显示 help 的用法。

3、向量与矩阵的表示及运算 变量也可用来存放向量或矩阵，并进行各种运算。

1) 向量的表示方法与运算 行向量的表示方法与运算示例如下:

```
x = [1 3 5 2]; %表示一个行向量
y = 2*x+1
y =
    3 7 11 5
```

可以随意更改、增加或删除向量中的元素：

```
y(3) = 2   % 更改第三个元素
y =        % 更改第三个元素后的结果
        3 7 2 5
y(6) = 10  % 加入第六个元素
y =        %加入第六个元素后的结果
        3 7 2 5 0 10
y(4) = []  % 删除第四个元素
y =        %删除第四个元素后的结果
         3 7 2 0 10
```

在上例中，百分比符号（%）为 MATLAB 中的注释符号，运行时 MATLAB 会忽略该符号后面的所有的文字。

MATLAB 亦可取出向量的一个元素或一部分向量来做运算：

```
x(2)*3+y(4)   % 取出 x 的第二个元素和 y 的第四个元素来做运算
ans =
        9
y(2:4)-1  % 用 y 的第二至第四个元素分别做减 1 运算，2:4 代表向量中的第 2、3、4 号元素
ans =
        6 1 -1
```

将行向量转置后，即可得到列向量：

```
z = x'
z =
    1
    3
```

5

2

2) 矩阵的表示方法和各种处理方式　一个三行四列的矩阵 A 在 MATLAB 中表示为:

A = [1 2 3 4; 5 6 7 8; 9 10 11 12]，在 MATLAB 的命令窗口中输入上述表达式，回车后得到下面的结果：

```
A =
     1    2    3    4
     5    6    7    8
     9   10   11   12
```

矩阵的运算示例如下：

```
A(2,3) = 5 %  将矩阵第二行，第三列的元素值置为 5
A =      %置值后的矩阵
   1   2   3   4
   5   6   5   8
   9   10 11   12
B = A(2,1:3) %取出矩阵 A 中第二行第一个到第三个元素，构成矩阵 B
B =
   5 6 5
A = [A B'] %  将 B 转置后，再以列向量并入 A
A =
   1   2   3    4   5
   5   6   5    8   6
   9   10 11   12   5
A(:, 2) = [] %  删除第二列，符号“：”代表所有列
A =
   1   3   4   5
   5   5   8   6
   9   11 12   5
A = [A; 4 3 2 1] %  加入第四行
A =
   1   3    4    5
   5   5    8    6
   9   11   12   5
   4   3    2    1
A([1 4], :) = [] %  删除第一和第四行，符号“:”代表所有行
```

```
A =
     5   5   8   6
     9  11  12   5
```

以上几种矩阵处理的方式可以相互叠加使用，产生各种意想不到的效果。在 MATLAB 中,每一个矩阵都是一个以行为主的阵列，因此对于矩阵元素的存取，可用一维或二维的下标值来确定。如在上述矩阵 A 中，位于第二列、第三行的元素可写为 A(2,3) （二维索引）或 A(6)（一维索引，即将所有直行进行堆叠后的第六个元素）。

2.1.3 MATLAB 绘图函数

在利用 MATLAB 中的神经网络工具箱创建神经网络时，二维绘图函数 plot 在显示神经网络的训练误差和网络的输出时经常用到。其基本调用格式有以下 3 种。

1、 plot(x) 当 x 为向量时，则以 x 元素为纵坐标，以相应元素的下标作为横坐标来绘图。当 x 为实数矩阵时，则按列绘制每列元素值相对其下标的连线图，图中曲线 x 阵的列数。

2、plot(x,y) 如果 x、y 为同维向量，则绘制以 x、y 为横纵坐标的连线图。如果 x 是向量，y 是一个与 x 同维的矩阵，则绘制多条不同色彩的连线图，连线条数等于 y 阵的另一维数。如果 x 和 y 是同维矩阵，则以 x、y 对应元素为横纵坐标分别绘制曲线，曲线的条数等于矩阵的行数。

3、 plot(x,y,s) s 表示线条的颜色和类型，如 s='r+'，表示各点是由红色的+号绘制的，如果没有特别说明，默认的类型为蓝色的线条。

此外与该函数联系比较密切的函数还有 hold on、hold off 和 figure，前两者用于在同一张图上绘制多条曲线的情况，后者用于停止绘制句柄，表示重新打开一张较长进行绘制。下面给出这几个函数的一个综合示例，绘制三条余弦曲线，其中第一张图上绘制两条，在第二张图上绘制一条。

在命令行窗口或在编辑调试窗口中输入代码，按回车键或 Run 命令即可得到图 2-1-3 和图 2-1-4 所示结果。完整的 MATLAB 代码如下：

```
%横坐标变化范围为[-6 6]，每间隔 0.1 个单位绘制一次
x=-6:0.1:6;
y1=cos(x);
y2=cos(2*x);
y3=cos(3*x);
%以 x、y 为横纵坐标绘图
plot(x,y1);
%保存绘图句柄，使下一次的图和已经绘制的图在同一张图上，如图 2-1-3 所示
hold on
plot(x,y2,'r+');
```

%关闭绘图句柄下次的图和已经绘制的图将不在现一张图上

hold off

%打开一张新的绘图面

Figure

%以 x、y 为横纵坐标，以蓝色的“*”绘图，如图 2-1-4 所示

plot(x,y3,'b*');

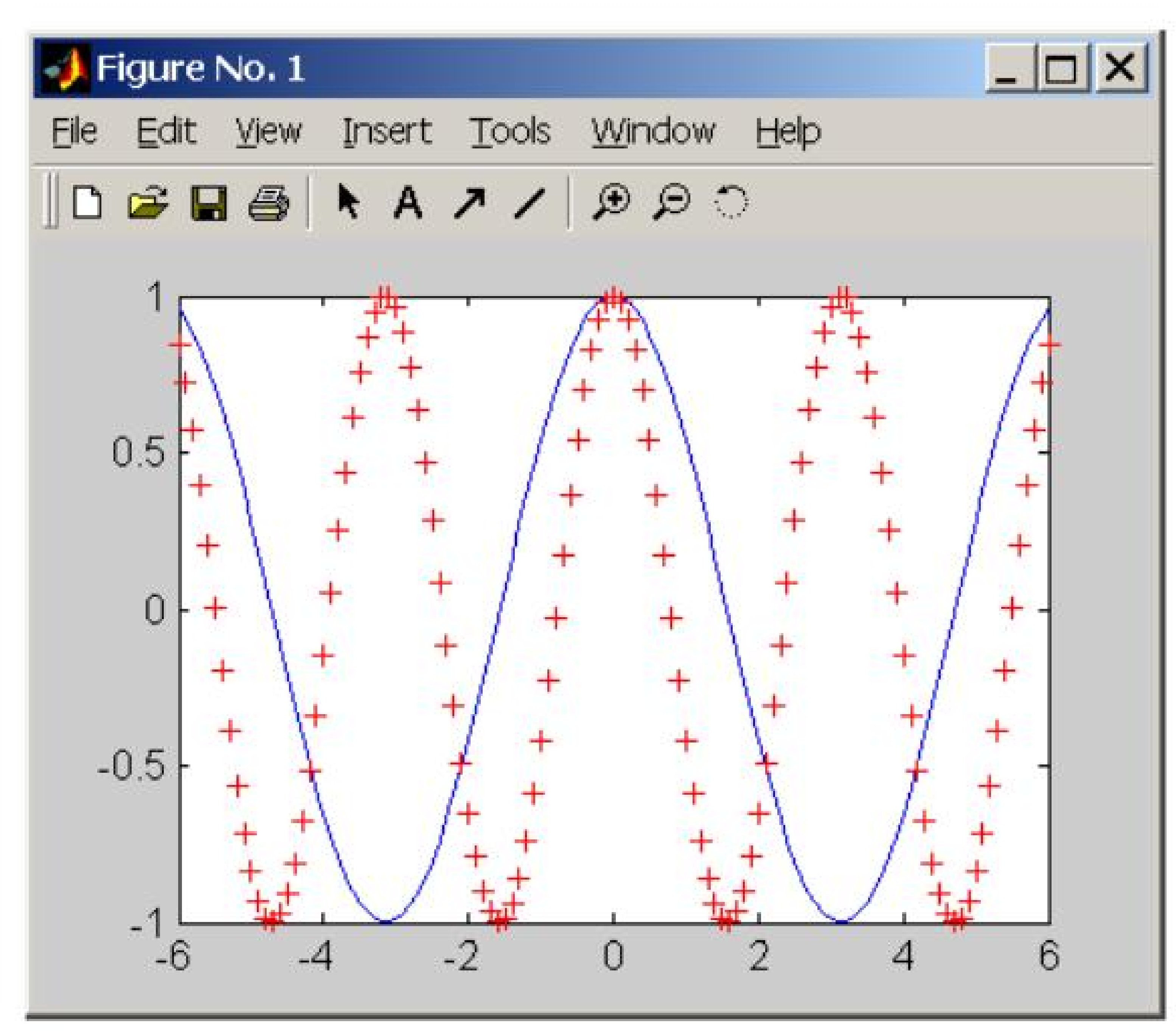


图 2-1-3 sinx 和 sin2x 的图形

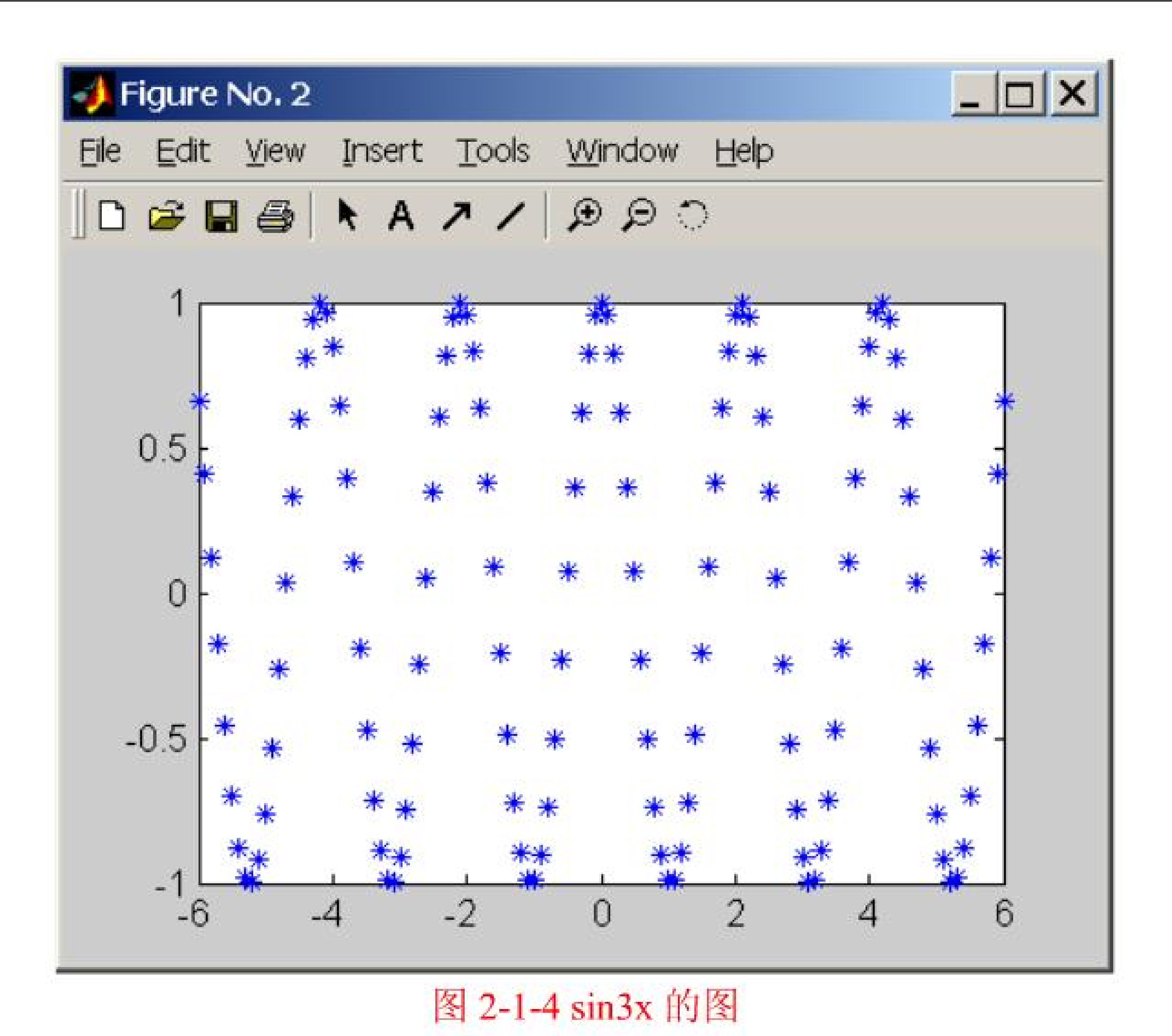


图 2-1-4 sin3x 的图

2.2 感知器神经网络模型与学习算法

2.2.1 单层感知器

感知器是由美国学者 Rosenblatt 在 1957 年首次提出的作为有导师学习(即有监督学习)的神经网络模型。单层感知器是指包含一个突触权值可调的神经元的感知器模型，它的学习算法是 Rosenblatt 在 1958 年提出的。感知器是神经网络用来进行模式识别的一种最简单模型，属于前向神经网络类型，但是仅由单个神经元组成的单层感知器只能用来实现线性可分的两类模式的识别。

单层感知器模型如图 2-2-1 所示，它包括一个线性的累加器和一个二值阈值元件，同时还有一个外部偏差 b ，也称作阈值，其值可以为正，也可以为负。线性累加器的输出与偏差 b 的和作为二值阈值元件的输入，这样当二值阈值元件的输入是正数，神经元就产生输出+1，反之，若输入是负

数，则产生输出-1。

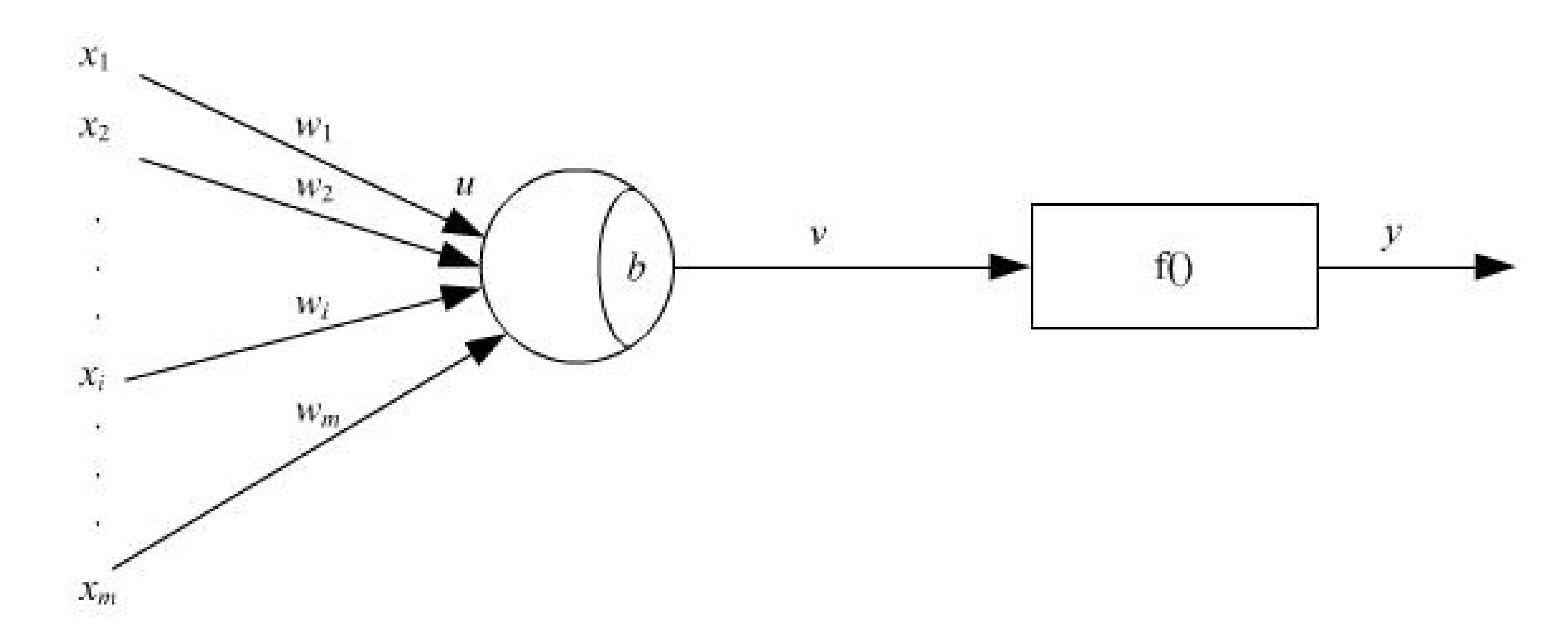


图2-1 单层感知器

使用单层感知器的目的就是让其对外部输入，$x_1, x_2, ..., x_m$ 进行识别分类，单层感知器可将外部输入分为两类 l_1 和 l_2。当感知器的输出为+1 时，输入 $x_1, x_2, ..., x_m$ 属于 l_1 类，当感知器的输出为-1 时，输入 $x_1, x_2, ..., x_m$ 属于 l_2 类，从而实现两类目标的识别。在 m 维空间，单层感知器进行模式识别的判决超平面由下式决定：

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

在图 2-2-2 中给出了一种只有两个输入下 x_1 和 x_2 时的判决超平面的情况，它的判别边界是直线：

$$w_1 x_1 + w_2 x_2 + b = 0$$

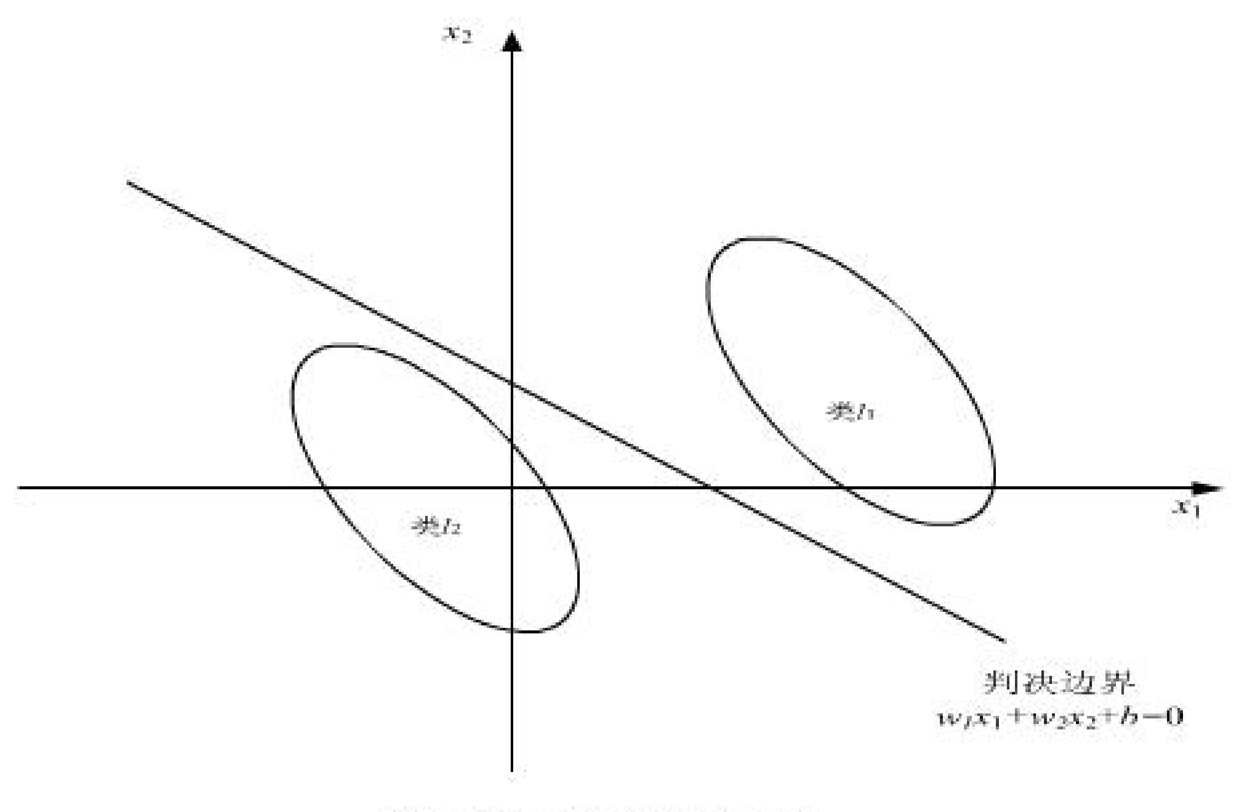


图2-2 两类模式识别的判定问题

决定判别边界直线形状的主要参数是权值向量 w_1 和 w_2，选择合适的学习算法可训练出满意的 w_1 和 w_2，当它用于两类模式的分类时，相当于在高维样本空间中，用一个超平面将两类样本分开。早在 20 世纪 60 年代初期，Rosenblatt 等就给出了严格的数学证明，对线性可分的样本，算法是一定收敛的，就是说 w 一定存在，否则，判别边界会产生振荡，导致 w 不能收敛。

2.2.2 单层感知器的学习算法

单层感知器的学习算法是基于迭代的思想，通常是采用误差校正学习规则的学习算法。为方便起见，将偏差 b 作为神经元突触权值向量的第一个分量加到权值向量中去，那么对应的输入向量也应增加一项，可设输入向量的第一个分量固定为+1，这样输入向量和权值向量可分别写成如下的形式：

$$\mathbf{X}(n)=\left[+1,x_1(n),x_2(n),\ldots,x_m(n)\right]^T$$

$$\mathbf{w}(n)=\left[b(n),w_1(n),w_2(n),\ldots,w_m(n)\right]^T$$

式中，n 表示迭代次数，其中的 $b(n)$ 可用 $\mathbf{W}^T(n)$ 表示，则二值阈值元件的输入可重新写为：

$$v=\sum_{i=0}^{m}w_i(n)x_i(n)=\mathbf{W}^T(n)\mathbf{X}(n)$$

令上式等于零，即 $\mathbf{W}^T(n)=0$ ，可得到在 m 维空间的单层感知器的判别超平面。考虑输出只有一个神经元的情况，具体学习算法如下：

第一步 设置变量和参量

$\mathbf{X}(n)=\left[1,x_1(n),x_2(n),\ldots,x_\mathrm{m}(n)\right]$ 为输入向量，也可以看成是训练样本。

$\mathbf{W}(n)=\left[b(n),w_1(n),w_2(n),\ldots,w_\mathrm{m}(n)\right]$ 为权值向量。

$b(n)$ 为偏差，$\mathrm{f}(\cdot)$ 为激活函数，$\mathbf{y}(n)$ 为网络实际输出，$d(n)$ 为期望输出，η 为学习速率，n 为迭代次数，e 为实际输出与期望输出的误差。

第二步 初始化 给权值向量 $\mathbf{W}(0)$ 的各个分量赋一个较小的随机非零值，置 $n=0$ 。

第三步 输入一组样本 $\mathbf{X}(n)=\left[1,x_1(n),x_2(n),\ldots,x_\mathrm{m}(n)\right]$，并给出它的期望输出 $d(n)$ 。

第四步 计算实际输出，

$$y(n)=\mathrm{f}\left(\sum_{i=0}^{m}w_i(n)x_i(n)\right)$$

第五步 求出期望输出和实际输出求出误差，$e=d(n)-y(n)$ ，根据误差判断目前输出是否满足条件，若满足条件则算法结束，否则将 n 值增加 1，并用下式调整权值：

$$w(n+1)=w(n)+\eta\left[d(n)-y(n)\right]\mathbf{X}(n)$$

然后转到第三步，进入下一轮计算过程。

注意：在以上学习算法的第五步需要判断是否满足算法结束条件，算法结束的条件可以是误差小于设定的值 ε 或者是权值的变化已很小。另外，在实现过程中还应设定最大的迭代次数，以防止算法不收敛时，学习算法进入死循环。

在单层感知器学习算法中，最关键的因素是引入了一个量化的期望输出，这样就可以采用误差校正学习规则对权值向量逐步进行修正，最终达到问题所需要的精度。

对于线性可分的两类模式，可以证明单层感知器的学习算法是收敛的，即通过调整神经网络各连接权值可以得到合适的判别边界，正确区分两类模式；而对于线性不可分的两类模式，无法用一条直线区分两类模式，此时，单层感知器的学习算法是不收敛的，即单层感知器无法正确区分线性不可分的两类模式。

2.2.3 单层感知器的 MATLAB 实现

MATLAB的神经网络工具箱中为单层感知器的设计、训练和学习等提供了丰富的工具函数，因篇幅的原因，只给出与本书所介绍的算法有关的工具函数，没有特别说明的话，后续章节也遵循此规定。与算法相关的感知器工具函数如表2-2-1所示。

表2-2-1　MATLAB中单层感知器常用工具函数名称和基本功能

函数名	功能
newp()	生成一个感知器
hardlim()	硬限幅激活函数
learnp()	感知器的学习函数
train()	神经网络训练函数
sim()	神经网络仿真函数
mae()	平均绝对误差性能函数
plotpv()	在坐标图上绘出样本点
plotpc()	在已绘制的图上加分类线

下面将对表 2-2-1 中的工具函数的使用进行说明，并通过一个实例来说明如何使用表中的工具函数建立一个对样本进行分类的感知器神经网络。

1.　Newp()

功能：创建一个感知器神经网络的函数

调用格式：net = newp(PR，S，TF，LF)

说明： net 为生成的感知机神经网络；PR 为一个 R×2 的矩阵，由 R 组输入向量中的最大值和最小值组成；S 表示神经元的个数；TF 表示感知器的激活函数，缺省值为硬限幅激活函数 hardlim；LF 表示网络的学习函数，缺省值为 learnp。

2.　Hardlim()

功能：硬限幅激活函数

格式：(1) A = hardlim(N)

说明：函数 hardlim(N)在给定网络的输入矢量矩阵 N 时，返回该层的输出矢量矩阵 A。当 N 中

的元素大于等于零时，返回的值为 l；否则为 0。也就是说，如果网络的输入达到阈值，则硬限幅传输函数的输出为 1；否则，为 0。

3. Learnp()

功能：感知机的权值和阈值学习函数

格式：(1) [dW，LS] = learnp(W，P，Z，N，A，T，E，gW，gA，D，LP，LS)

(2) [db，LS] = learnp(b，ones(1，Q)，Z，N，A，T，E，gW，gA，D，LP，LS)

说明： dW 为权值变化矩阵；LS 为当前学习状态；W 为 S×R 的权值矩阵(可省略)；P 为 R×Q 的输入向量矩阵； Z 为 S×Q 的输入层的权值矩阵(可省略)；N 为 S×Q 的网络输入矩阵(可省略)；A 为 S×Q 的输出矩阵(可省略)；T 为 S×Q 的目标输出矩阵(可省略)；E 为误差向量(E=T-Y)，T 表示网络的目标向量(可省略)；Y 表示网络的实际输出向量(可省略)；gW 为 S×R 的与性能相关的权值梯度矩阵(可省略)；gA 为 S×Q 的与性能相关的输出梯度值矩阵(可省略)；D 为 S×S 的神经元距离矩阵(可省略)；LP 为学习参数(可省略)；LS 学习函数声明(可省略)；b 为 S×1 的阈值矢量；ones(1，Q)为 1×Q 的全为 1 的向量。

3. Train()

功能：神经网络训练函数

格式：[net，tr，Y，E，Pf，Af] = train(NET，P，T，Pi，Ai，VV，TV)

说明：式中，net 为训练后的网络；tr 为训练记录；Y 为网络输出矢量；E 为误差矢量；Pf 为训练终止时的输入延迟状态；Af 为训练终止时的层延迟状态；NET 为训练前的网络；P 为网络的输入向量矩阵；T 表示网络的目标矩阵，缺省值为 0；Pi 表示初始输入延时，缺省值为 0；Ai 表示初始的层延时，缺省值为 0； VV 为验证矢量（可省略）；TV 为测试矢量（可省略）。网络训练函数是一种通用的学习函数，训练函数重复地把一组输入向量应用到一个网络上，每次都更新网络，直到达到了某种准则，停止准则可能是达到最大的学习步数、最小的误差梯度或误差目标等。

4. Sim()

功能：对网络进行仿真

格式：(1) [Y，Pf，Af，E，perf] = sim(NET，P，Pi，Ai，T)

(2) [Y，Pf，Af，E，perf] = sim(NET，{Q TS}，Pi，Ai，T)

(3) [Y，Pf，Af，E，perf] = sim(NET，Q，Pi，Ai，T)

说明：式中 Y 为网络的输出；Pf 表示最终的输入延时状态；Af 表示最终的层延时状态；E 为实际输出与目标矢量之间的误差；perf 为网络的性能值；NET 为要测试的网络对象；P 为网络的输入向量矩阵；Pi 为初始的输入延时状态（可省略）；Ai 为初始的层延时状态（可省略）；T 为目标矢量（可省略）。式(1)、(2)用于没有输入的网络，其中 Q 为批处理数据的个数，TS 为网络仿真的时间步数。

5. Mae()

功能：平均绝对误差性能函数

格式：(1) perf=mae(E，w，pp)

说明: perf 表示平均绝对误差和， E 为误差矩阵或向量(网络的目标向量与输出向量之差)， w 为所有权值和偏值向量(可忽略)， pp 为性能参数(可忽略)。

6. Plotpv()

功能：绘制样本点的函数

格式：(1) plotpv(P，T)

(2) plotpv(P，T，V)

说明：式中，P 定义了 n 个 2 或 3 维的样本，是一个 2×n 维或 3×n 维的矩阵；T 表示各样本点的类别，是一个 n 维的向量；V=[x_min x_max y_min y_max]，为一设置绘图坐标值范围的向量。利用 plotpv()函数可在坐标图中绘出给定的样本点及其类别，不同的类别使用不同的符号。如果 T 只含一元矢量，则目标为 0 的输入矢量在坐标图中用符号“o”表示： 目标为 1 的输入矢量在坐标图中用符号“+”表示。如果 T 含二元矢量，则输入矢量在坐标图中所采用的符号分别如下：[0 0]用“o”表示；[0 1]用“+”表示：[1 0]用“*”表示；[1 1]用“×”表示。

7.Plotpc()

功能：在存在的图上绘制出感知器的分类线函数

格式：(1) plotpc(W，B)

(2) plotpc(W，B，H)

说明：硬特性神经元可将输入空间用一条直线(如果神经元有两个输入)，或用一个平面(如果神经元有三个输入)，或用一个超平面(如果神经元有三个以上输入)分成两个区域。plotpc(w，b)对含权矩阵 w 和偏差矢量 b 的硬特性神经元的两个或三个输入画一个分类线。这一函数返回分类线的句柄以便以后调用。plotpc(W，B，H)包含从前的一次调用中返回的句柄。它在画新分类线之前，删除旧线。

使用 MATLAB 实现神经网络的步骤如下：

第一步 根据应用创建一个神经网络；

第二步 设定神经网络的训练参数，利用给定样本对创建的神经网络进行训练；

第三步 输入测试数据，测试训练好的神经网络的性能；

下面的例子 2-1 是一个根据神经网络 MATLAB 实现的步骤，并使用上面介绍的感知器神经网络的 MATLAB 工具函数，建立一个感知器神经网络，设定训练参数，然后使用样本数据对网络进行训练，训练过程是通过感知器的学习算法调整权值和阈值，直到达到预先设置的训练要求为止，同时给出训练后的感知器的性能，最后使用测试数据对训练好的神经网络进行测试，并给出测试的结果。

例子 2-1 从待分类的数据中取出一部分数据及其对应的类别做为样本数据，设计并训练一个能对待分类数据进行分类的单层感知器神经网络。

感知器神经网络的 MATLAB 代码如下：

```
%给定训练样本数据
P= [-0.4 -0.5 0.6; 0.9 0 0.1];
%给定样本数据所对应的类别，用 1 和 0 来表示两种类别
T= [1 1 0];
%创建一个有两个输入、样本数据的取值范围都在[-1，1]之间，并且网络只有一个神经元的感
%知器神经网络
net=newp([-1 1;-1 1],1);
%设置网络的最大训练次数为 20 次，即训练 20 次后结束训练
net.trainParam.epochs = 20;
%使用训练函数对创建的网络进行训练
net=train(net,P,T);
%对训练后的网络进行仿真，即根据训练后的网络和样本数据给出输出
Y=sim(net,P)
%计算网络的平均绝对误差，表示网络错误分类
E1=mae(Y-T)
%给定测试数据，检测训练好的神经网络的性能
Q=[0.6 0.9 -0.1; -0.1 -0.5 0.5];
%使用测试数据，对网络进行仿真，仿真输出即为分类的结果
Y1=sim(net,Q)
%创建一个新的绘图窗口
figure;
%在坐标图中绘制测试数据点，并根据数据所对应的类别用约定的符号画出
plotpv(Q,Y1);
%在坐标图中绘制分类线
plotpc(net.iw{1},net.b{1})
```

运行后得到图 2-2-3 和图 2-2-4，在命令行窗口中得到的结果如下：

```
%使用 TRAINC 作为神经网络的训练函数，第 0 次训练，最大训练次数为 20
>> TRAINC, Epoch 0/20
TRAINC, Epoch 3/20
%达到目标误差要求，结束训练
TRAINC, Performance goal met.

Y =
     1     1     0
```

E1 =

0

Y1 =

0　　0　　1

由图 2-2-3 和命令行窗口中的数据可知，网络的训练过程只经过 3 次，就达到了目标性能要求，网络的平均绝对误差已经为 0，从图 2-2-4 中的分类线和命令行窗口中的 Y1 =　0　　0　　1 可以看出创建的感知器网络成功地将测试数据分为两类。

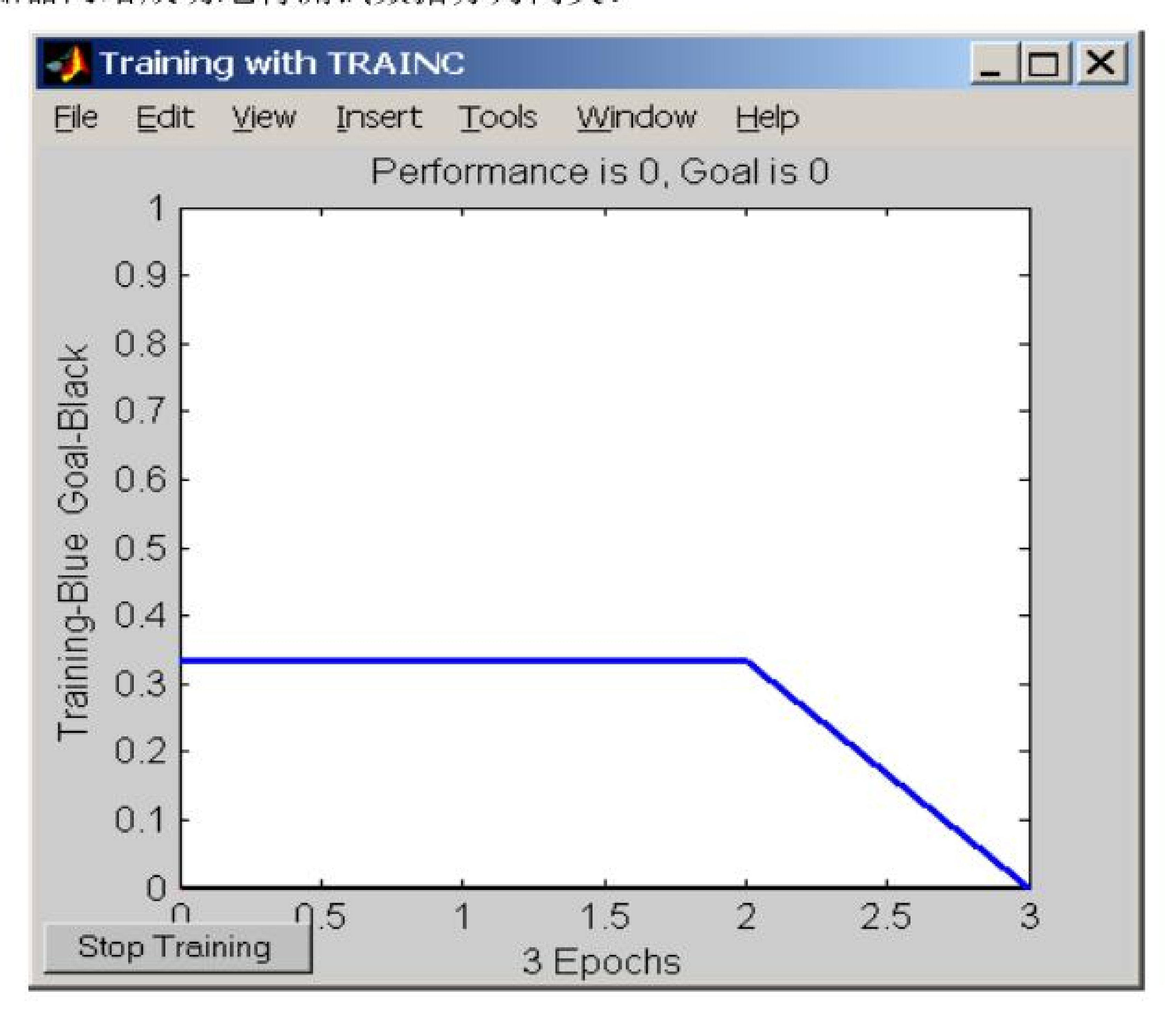


图 2-2-3

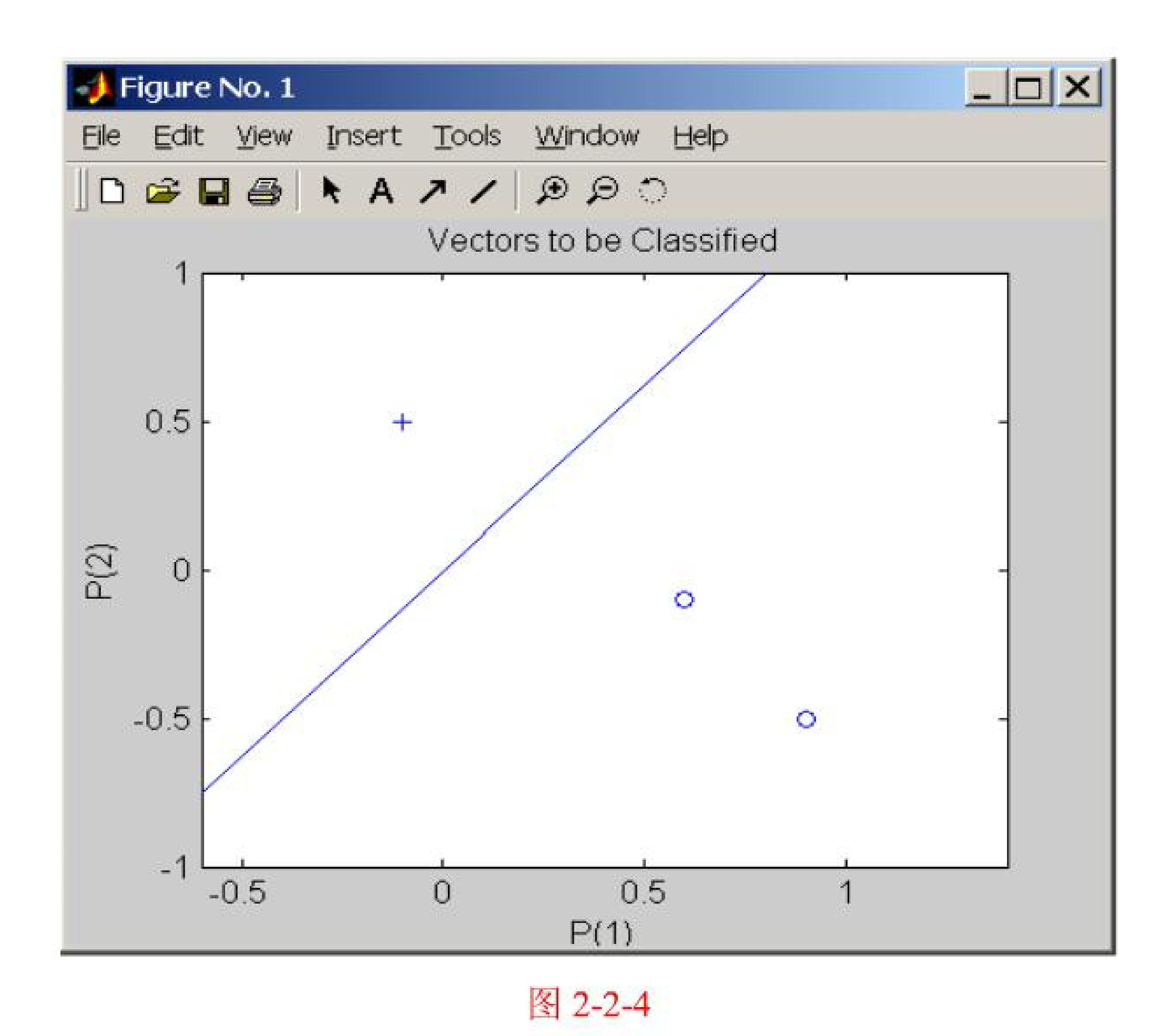
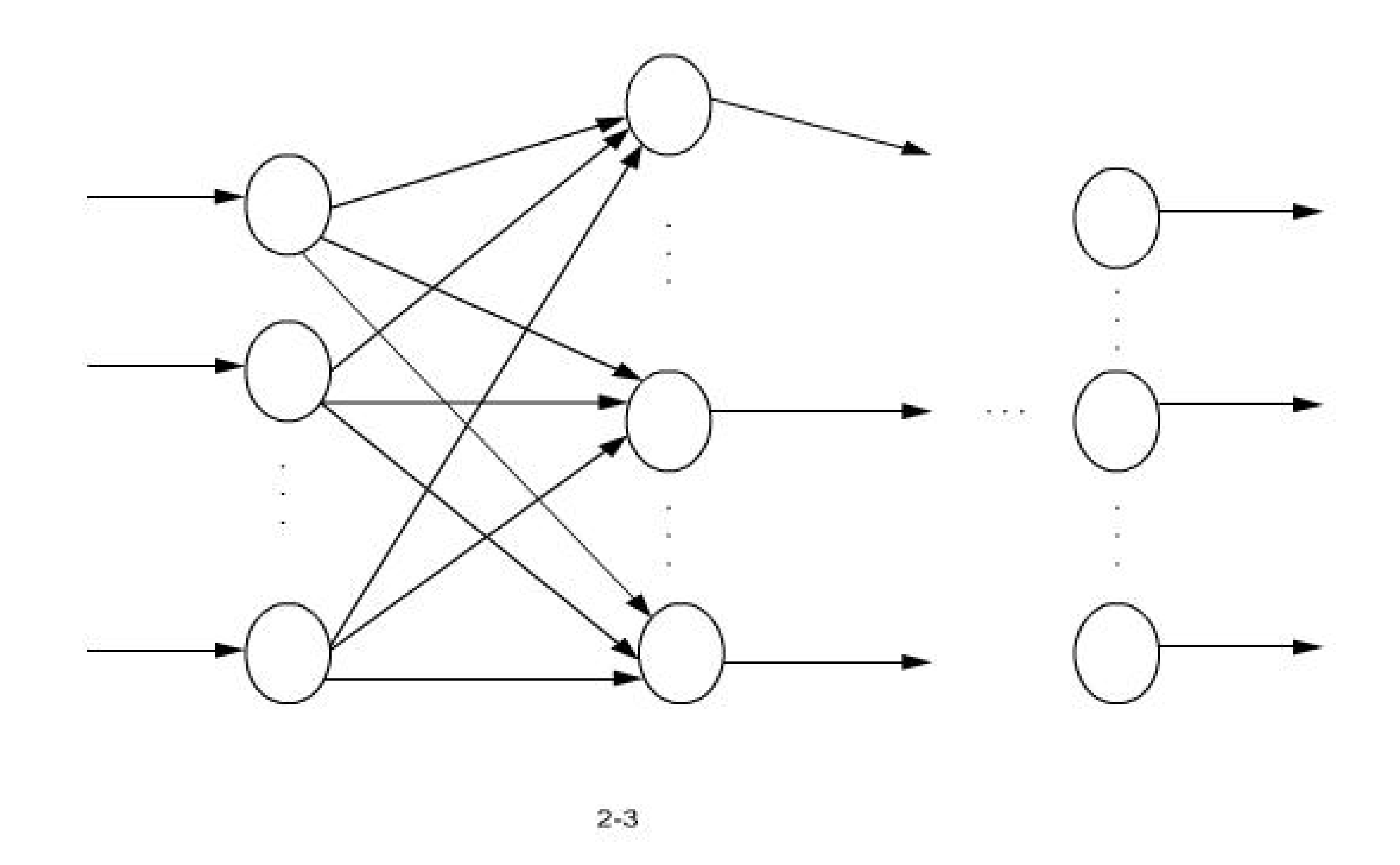


图 2-2-4

2.2.4 多层感知机

前面介绍了单层感知器，它最大的缺点是只能解决线性可分的分类模式问题，要增强网络的分类能力唯一的方法是采用多层网络结构，即在输入层与输出层之间增加一个隐含层，从而构成多层感知器(Multilayer Perceprons ，MLP)。这种由输入层、隐含层(可以是一层或者多层)和输出层构成的神经网络称为多层前向神经网络。

多层感知器是对单层感知器的推广，它能够成功解决单层感知器所不能解决的非线性可分问题。其拓扑结构如图 2-2-5 所示。



2-3

输入层神经元的个数等于输入信号的个数，隐含层个数以及隐含层结点的个数则视具体情况而定，输出层神经元的个数为输出信号的个数。

多层感知器同单层感知器相比具有四个明显的特点：

1)多层感知器含有一层或多层隐单元 隐单元从输入模式中获得了更多有用的信息，使网络可以完成更复杂的任务。

2)多层感知器中每个神经元的激活函数采用可微的函数，如：sigmoid 函数：

$$v_i = \frac{1}{1+\exp(-u_i)}$$

式中 u_i 是第 i 个神经元的输入信号，v_i 是该神经元的输出信号。

3)多层感知器的多个突触使得网络更具连通性，连接域的变化以及连接权值的变化都会引起连通性的变化。

4)多层感知器具有独特的学习算法，该学习算法就是著名的 BP 算法，因此使用 BP 算法的神经网络也常常被称为 BP 网络。

多层感知器所具有的这些特点，使得它具有强大的计算能力从而成为一种广泛使用的神经网络。关于多层感知器的学习算法即 BP 算法，将在 2.3 节中详细介绍。

2.3 线性神经网络模型与学习算法

线性神经网络类似于感知器，但是线性神经网络的激活函数是线性的，而不是硬限转移函数。因此，线性神经网络的输出可以是任意值，而感知器的输出不是 0 就是 1。线性神经网络和感知器一样只能求解线性可分的问题。因此，线性神经网络的限制和感知器相同。

2. 3. 1 线性神经元网络模型

1.线性神经元模型

具有 m 个输入的线性神经元如图 2-3-1 所示。线性神经元与感知器神经元具有相似的结构，唯一的不同是线性神经元使用了线性传递函数 purelin，因此与感知器神经网络不同，线性神经网络的输出可以取任意值，而不像感知器神经网络的输出只能是 0 或 1，它的形状如图 2-3-2 所示。

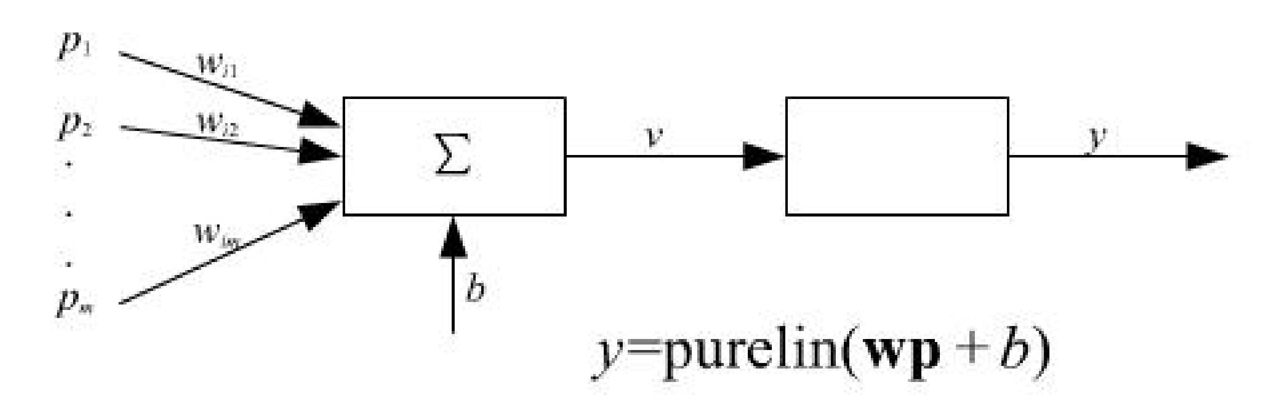


图2-4 线性神经元模型

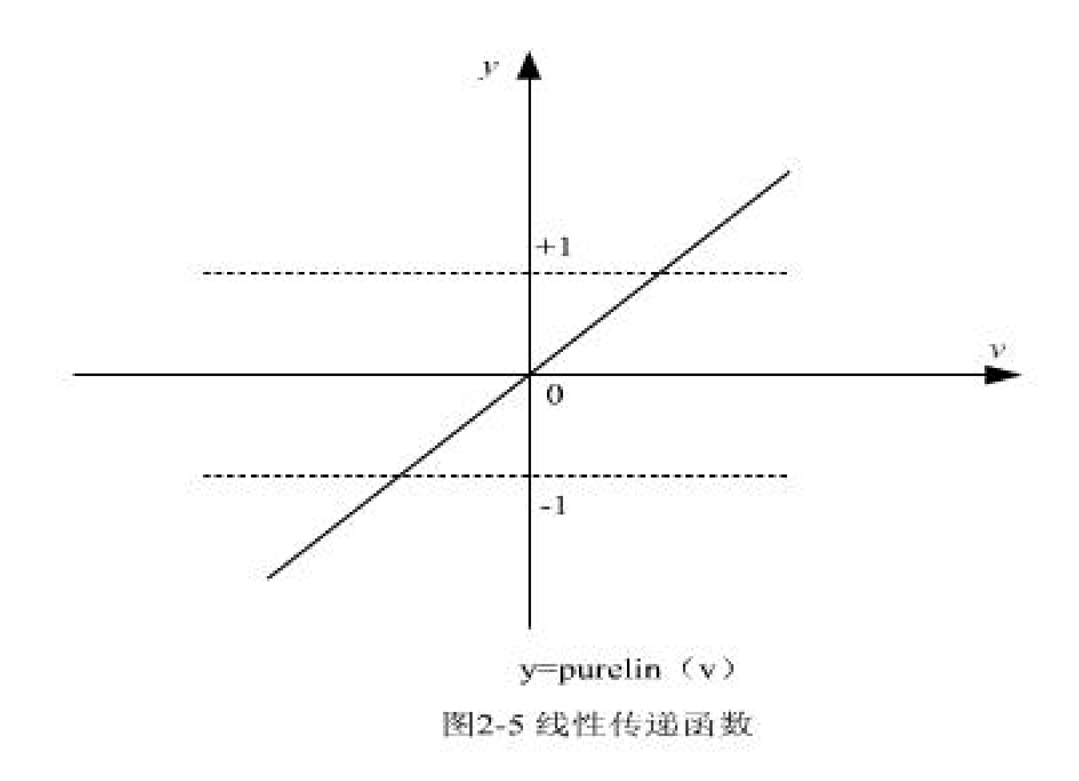


图2-5 线性传递函数

线性神经元的输出可以由以下公式计算：

$$y = \text{purelin}(v) = \text{purelin}(\mathbf{wp}+b) = \mathbf{wp}+b$$

现在，考虑一个具有两个输入的线性神经元，如图 2-6 所示，网络的输出为

$$y = w_{11}p_1 + w_{21}p_2 + b$$

类似于感知器神经网络，它也可以区分两类输入向量。当输出 y 等于 0 时，可以画出它们的分界线，如图 2-3-3 所示。

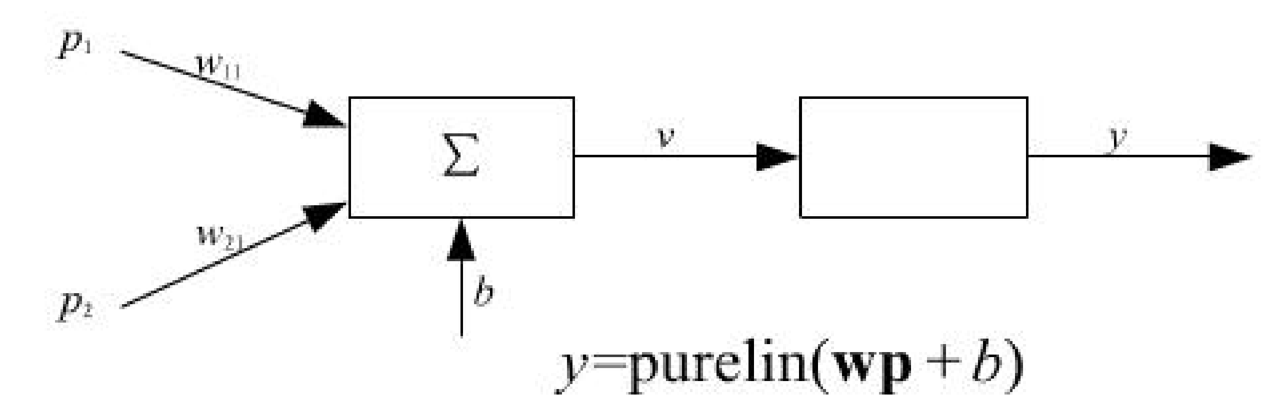


图2-6 双输入线性神经元

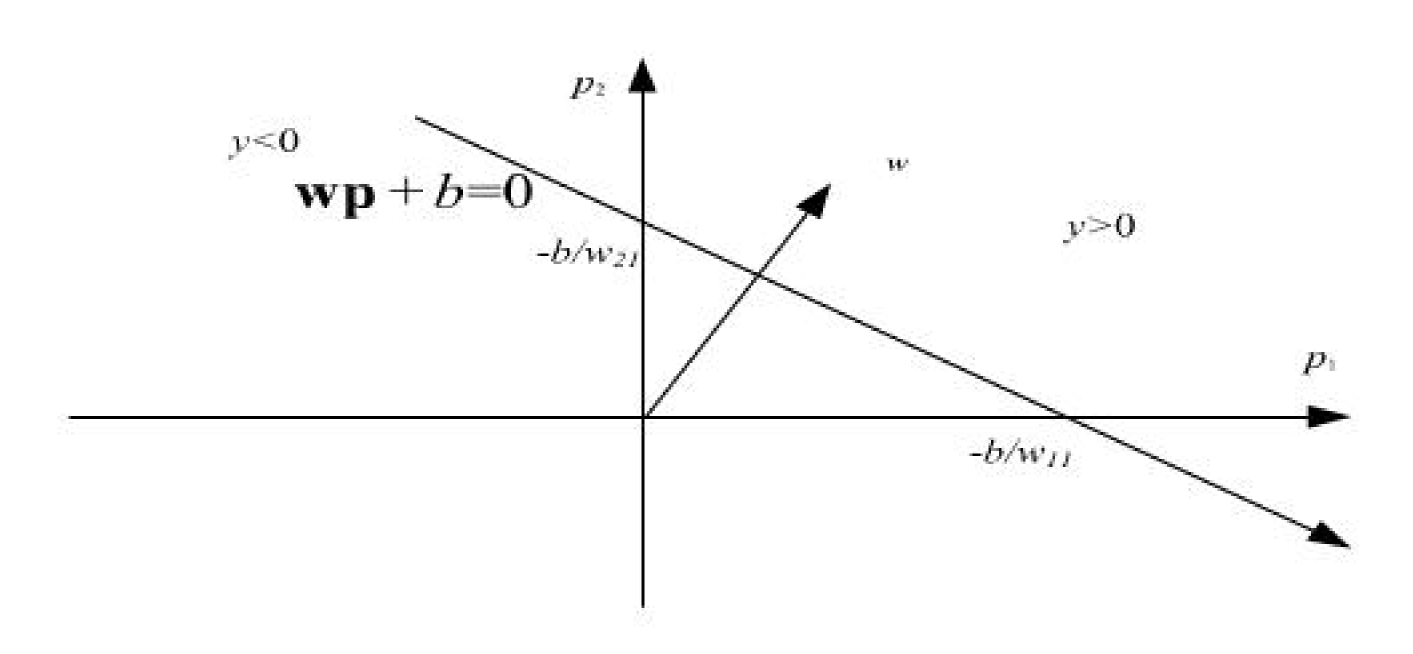


图2-7 输入向量分类结果

位于分界线上面的输入向量能够产生大于 0 的网络输出，位于分界线下面的输入向量则产生小于 0 的网络输出。因此线性神经元只能逼近一个线性函数，而不能完成逼近非线性函数的计算，其局限性与感知器神经网络相同。

2.线性神经网络的结构

图 2-3-4 显示了单层线性神经网络的结构，其中网络具有 R 个输入，S 个神经元，并通过权值 w_{ij} 连接，下标 i 和 j 表示权值 w_{ij} 连接着第 j 个神经元和第 i 个输入，通过网络后产生 S 个输出。

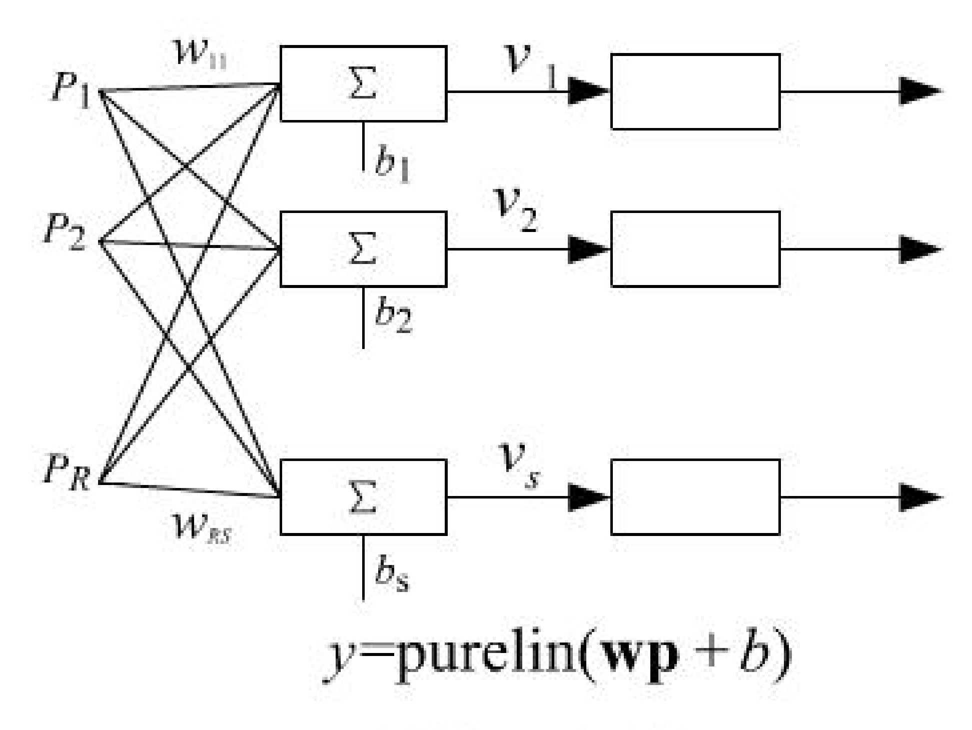


图2-8 线性神经网络结构

2.3.2 线性神经网络的学习算法

线性神经网络采取的学习规则是 Widrow-Hoff 学习规则，又称为最小均方误差 LMS（Least Mean Square Error）学习算法，它是基于负梯度下降的原则来减小网络的训练误差。最小均方误差学习算法也属于监督类学习算法。令 $\mathbf{p}_k=(p_1(k),p_2(k),\cdots$ 表示网络的输入向量，$\mathbf{d}_k=(d_1(k),d_2(k),\cdots$ 表示网络的期望输出向量，$\mathbf{y}_k=(y_1(k),y_2(k),\cdots$ 表示网络的实际输出向量。其中 $k=1,2,\cdots$ ，表示输入向量与对应的期望输出向量样本对的数量，计算出实际输出向量与相应的期望输出向量的误差，并且依据误差来调整网络的权值和阈值，使该误差逐渐减小。LMS 学习规则就是要减小这些误差平方和的均值，定义如下：

$$mse=\frac{1}{m}\sum_{k=1}^{m}e^2(k)=\frac{1}{m}\sum_{k=1}^{m}\left(d(k)-y(k)\right)^2$$

从最小均方误差的定义可以看出，它的性能指标是一个二次方程，所以它要么具有全局最小值，要么没有最小值，而选择什么样的输入向量恰恰会决定网络的性能指标会有什么样的最小值。

如果考虑第 k 次循环时训练误差的平方对网络权值和阈值的二阶偏微分，会得到公式：

$$\frac{\partial e^2(k)}{\partial w_{ij}} = 2e(k)\frac{\partial e\ (k)}{\partial w_{ij}}$$

其中 $j = 1,2,\cdots$

$$\frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e\ (k)}{\partial b}$$

再计算此时的训练误差对网络权值和阈值的一阶偏微分，

$$\frac{\partial e(k)}{\partial w_{ij}} = \frac{\partial\left[d(k) - y(k)\right]}{\partial w_{ij}} = \frac{\partial e}{\partial w_{ij}}\left[d(k) - \left(\mathbf{Wp}(k) + b\right)\right]$$

或者

$$\frac{\partial e(k)}{\partial w_{ij}} = \frac{\partial e}{\partial w_{ij}}\left[d(k) - \left(\sum_{i=1}^{R} w_{ij} p_i(k) + b\right)\right] \qquad j = 1,2,\cdots$$

其中 $p_i(k)$ 表示第 k 次循环中的第 i 个输入向量。则有

$$\frac{\partial e(k)}{\partial w_{ij}} = -p_i(k)$$

$$\frac{\partial e(k)}{\partial b} = -1$$

那么根据负梯度下降的原则，网络权值和阈值的改变量应该是 $2\eta e(k)\mathbf{p}(k)$ 和 $2\eta e(k)$
所以网络权值和阈值修正公式如下：

$$w(k+1) = w(k) + 2\eta e(k)\mathbf{p}^T(k)$$
$$b(k+1) = b(k) + 2\eta e(k)$$

式中，η 是学习率。当 η 取较大值时，可以加快网络的训练速度，但是如果 η 的值太大，会导致网络稳定性的降低和训练误差的增加。所以，为了保证网络进行稳定的训练，学习率 η 的值必须

选择一个合适的值。

重复以上求解过程，直到达到预定的精度，算法结束。

2.3.3 线性神经网络的 MATLAB 实现

MATLAB 神经网络工具箱中提供的与算法相关的线性神经网络相关的工具箱函数，如表 2-3-1 所示。在 MATLAB 的命令行窗口中输入“help linnet"，便可得到与线性网络相关的所有函数，进一步利用 help 命令又能得到相关函数的详细介绍

表 2-3-1　MATLAB 提供的线性神经网络的常用函数和基本功能表

函数名	功能
newlin()	新建一个线性层
learnwh()	Widrow-Hoff 的学习函数
purelin()	线性传输函数
mse()	最小均方误差性能函数

下面将对表 2-3-1 中的工具函数的使用进行说明，并通过一个实例来说明如何使用表中的工具函数建立一个线性神经网络。

1. Newlin()

功能：新建一个线性神经网络函数。

格式：(1)　net = newlin

(2)　net = newlin(PR，S，ID，LR)

说明：式(1)返回一个没有定义结构的空对象，并显示图形用户界面函数 nntool 的帮助文字；式(2)中 net 为生成的线性神经网络；PR 为网络输入向量中的最大值和最小值组成的矩阵[Pmin，Pmax]；S 为输出向量的个数；ID 为输入延时向量（可省略）；LR 为学习速率（可省略），默认值为 0.01。线性神经网络主要用于自适应滤波器设计和信号预测，该网络由一层神经元组成，神经元的加权函数为 dotprod，输入函数为 netsum，传递函数为 purelin，神经元的权值和阈值初始化函数为 initzero，自适应调整函数为 trains，训练函数为 trainb，它们都使用学习函数 learnwh 对权值和阈值进行调整，性能函数为 mse。

2. Learnwh( )

功能：　线性神经网络学习函数

格式： (1) [dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

(2) [db,LS] = learnwh(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)

说明： 式中，dW 为权值变化矩阵；LS 为阈值变化阵；W 为 S×R 的权值矩阵(可省略)；P 为 R×Q 的输入向量矩阵或者为 1×Q 的全为 1 的向量；Z 为 S×Q 的输入层的权值矩阵(可省略)；N 为 S×Q 的网络输入矩阵(可省略)；A 为 S×Q 的输出矩阵(可省略)；T 为 S×Q 的目标输出矩阵(可省略)；E 为误差向量(E=T-Y)，T 表示网络的目标向量(可省略)；Y 表示网络的实际输出向量(可省略)；gW 为 S×R 的与性能相关的权重梯度矩阵(可省略)；gA 为 S×Q 的与性能相关的输出梯度值矩阵(可省略)；D 为 S×S 的神经元距离矩阵(可省略)；LP 为学习参数(可省略)；LS 学习函数声明(可省略)。

learnwh()是利用 Widrow-hoff 规则对网络的权值和阈值进行学习，该函数多用于线性网络。Widrow-Hoff 学习规则只能训练单层的线性神经网络，但这并不影响单层线性神经网络的应用，因为对每一个多层线性神经网络而言，都可以设计出一个性能完全相当的单层线性神经网络

3. Purelin()

功能：纯线性传输函数

格式： A = purelin(N)

说明：函数 purelin(N)为返回网络输入向量 N 的输出矩阵 a；神经元最简单的传输函数是简单地从神经元输入到输出的线性传输函数，输出仅仅被神经元所附加的偏差所修正，newlin 和 newlind 函数建立的网络都可以用该函数做为传递函数。

4. mse()

功能：均方误差性能函数

格式：(1) perf=mae(E，w，pp)

说明: perf 表示均方误差， E 为误差矩阵或向量(网络的目标向量与输出向量之差)， w 为所有权值和偏值向量(可忽略)， pp 为性能参数(可忽略)。

例子 2-2 要求设计一个线性神经网络，寻找给定数据之间的线性关系。

```
P=[1.1 -1.3];
T=[0.6 1];
%创建一个只有一个输出，输入延时为 0，学习速率为 0.01 的线性神经网络，minmax(P)表示样
%本数据的取值范围
net=newlin(minmax(P),1,0,0.01);
%对创建的线性神经网络进行初始化，设置权值和阈值的初始值
net=init(net);
net.trainParam.epochs=500;
%设置网络训练后的目标误差为 0.0001
net.trainParam.goal=0.0001;
net=train(net,P,T);
```

```
y=sim(net,P)
%求解网络的均方误差值
E=mse(y-T)
```

运行后得到图 2-3-5，在命令行窗口中得到的结果如下：

```
%使用 TRAINB 作为训练函数，最大训练次数为 500，开始训练时的均方误差值为 0.68，
%目标误差为 0.0001
>> TRAINB, Epoch 0/500, MSE 0.68/0.0001.
TRAINB, Epoch 25/500, MSE 0.230041/0.0001.
TRAINB, Epoch 50/500, MSE 0.0804376/0.0001.
TRAINB, Epoch 75/500, MSE 0.0287728/0.0001.
TRAINB, Epoch 100/500, MSE 0.0104466/0.0001.
TRAINB, Epoch 125/500, MSE 0.00382901/0.0001.
TRAINB, Epoch 150/500, MSE 0.00141177/0.0001.
TRAINB, Epoch 175/500, MSE 0.000522418/0.0001.
TRAINB, Epoch 200/500, MSE 0.000193748/0.0001.
TRAINB, Epoch 217/500, MSE 9.87777e-005/0.0001.
%训练到 217 次时，达到目标误差要求，结束训练
TRAINB, Performance goal met.

y =
      0.5883        0.9922
E =
   9.8778e-005
```

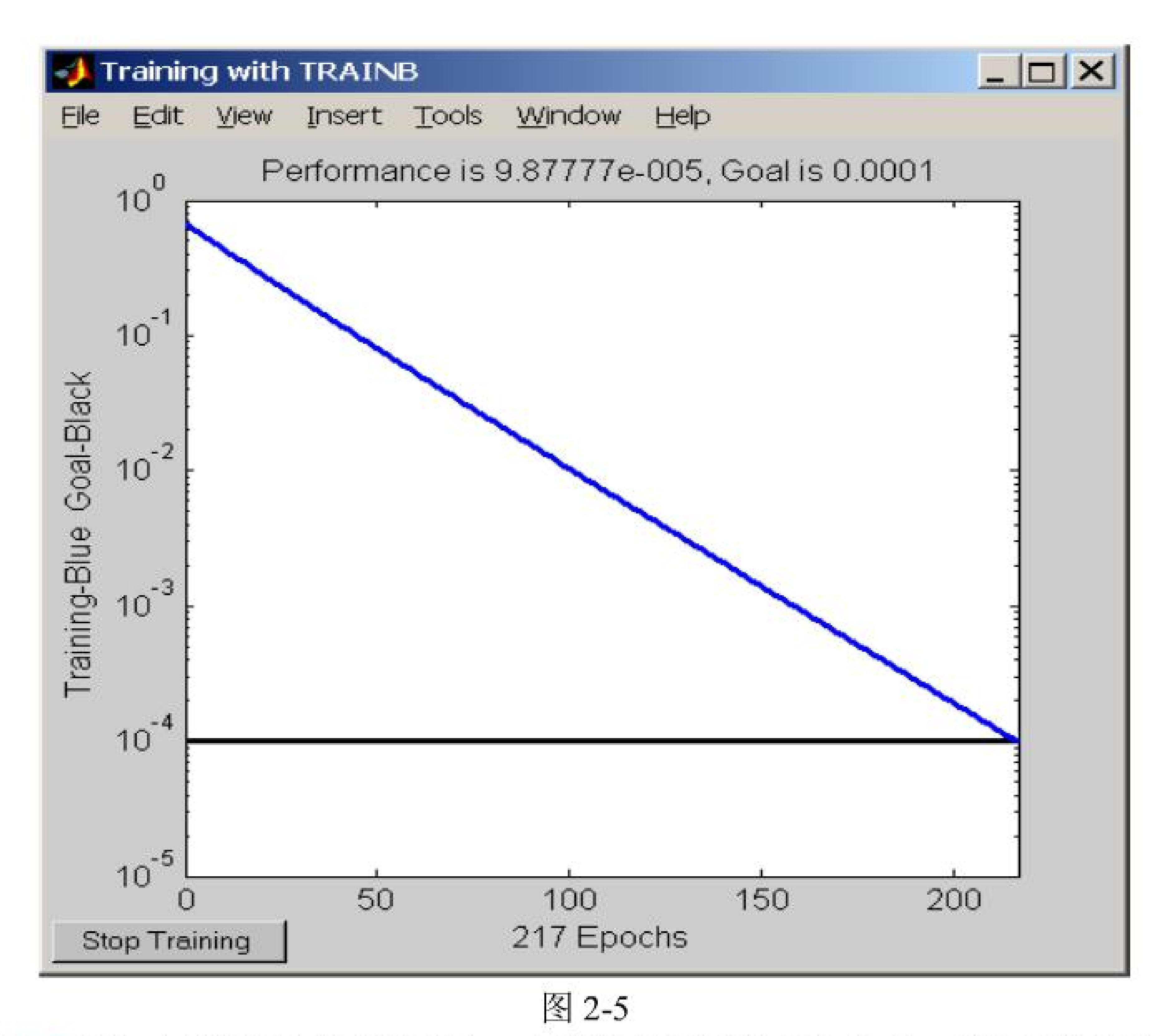


图 2-5

由图 2-3-5 和命令行窗口中的数据可知，网络的训练经过 217 次后，达到了预定的目标性能要求，网络的均方误差值已经为 0.0001。

2.4 BP 神经网络模型与学习算法

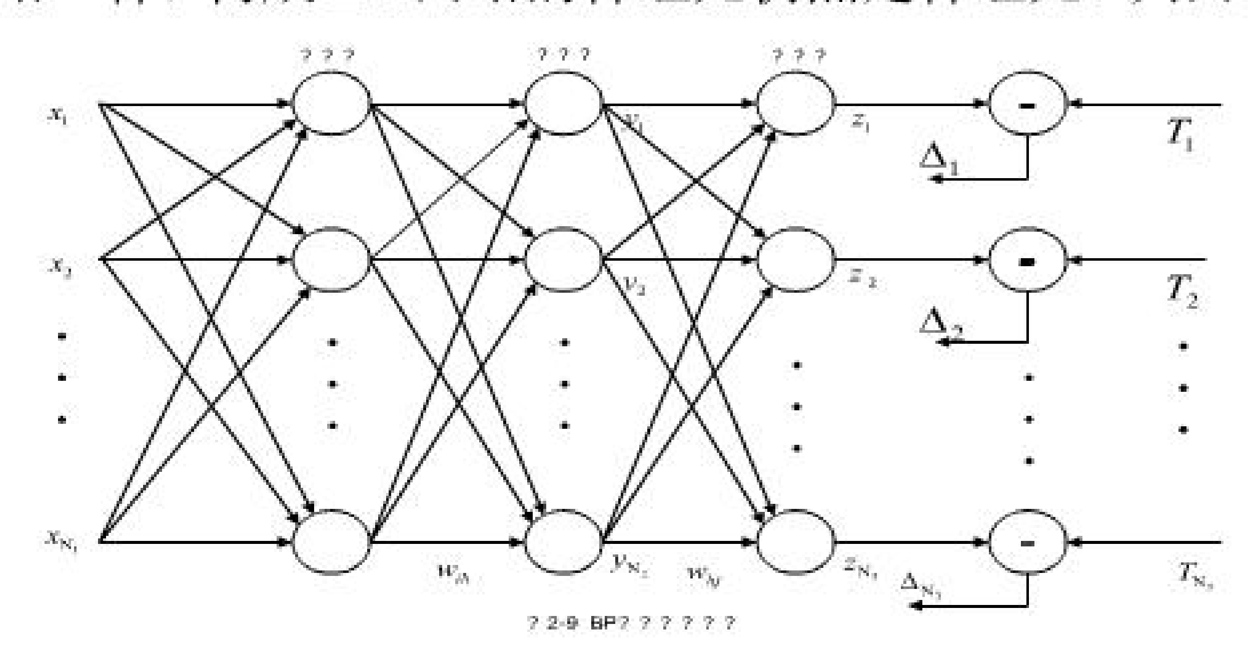
在前面介绍的感知器和线性神经网络的学习算法中，理想输出与实际输出之差被用来估计神经元连接权值的误差。当为解决线性不可分问题而引入多级网络后，如何估计网络隐含层神经元的误差就成了一大难题。因为在实际中，无法知道隐含层的任何神经元的理想输出值。Rumelhart，McClelland 和他们的同事们洞察到了神经网络信息处理的重要性，并于 1982 年成立了一个 PDP 小组，研究并行分布式信息处理方法，探索人类认知的微结构。1985 年他们提出了 BP 网络的误差反向后传 BP(Back Propagation)学习算法，实现了 Minsky 设想的多层神经网络模型。

BP(Back Propagation)算法在于利用输出后的误差来估计输出层的直接前导层的误差，再用这个

误差估计更前一层的误差，如此一层一层的反传下去，就获得了所有其它各层的误差估计。这样就形成了将输出层表现出的误差沿着与输入传送相反的方向逐级向网络的输入层传递的过程。因此，人们特将此算法称为误差反向后传算法，简称 BP 算法。使用 BP 算法进行学习的多级非循环网络称为 BP 网络，属于前向神经网络类型。虽然这种误差估计本身的精度会随着误差本身的“向后传播”而不断降低，但它还是给多层网络的训练提供了比较有效的办法，加之多层前向神经网络能逼近任意非线性函数，在科学技术领域中有广泛的应用，所以，多年来该算法一直受到人们广泛的关注。

2.4.1BP 神经网络模型

与一般的人工神经网络一样，构成 BP 网络的神经元仍然是神经元。其网络模型如图 2-4-1 所示



? 2-9 BP? ? ? ? ? ?

按照 BP 算法的要求，这些神经元所用的激活函数必须是处处可导的。一般都使用 S 型函数。对一个神经元来说，它的网络输入可表示为

$$net = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

其中，$x_1, x_2, \cdots$ 为该神经元所接受的输人，$w_1, w_2, \cdots$ 分别是它们对应的连联接权值。该神经元的输出为：

$$y = \mathrm{f}(net) = \frac{1}{1 + \mathrm{e}^{-net}}$$

其相应的图像如图 2-4-2 所示。当 net=0 时，y 取值为 0.5，并且 net 落在区间(-0.6，0.6)中时，y 的变化率较大，而在（-1，1）之外，y 的变化率就非常小。

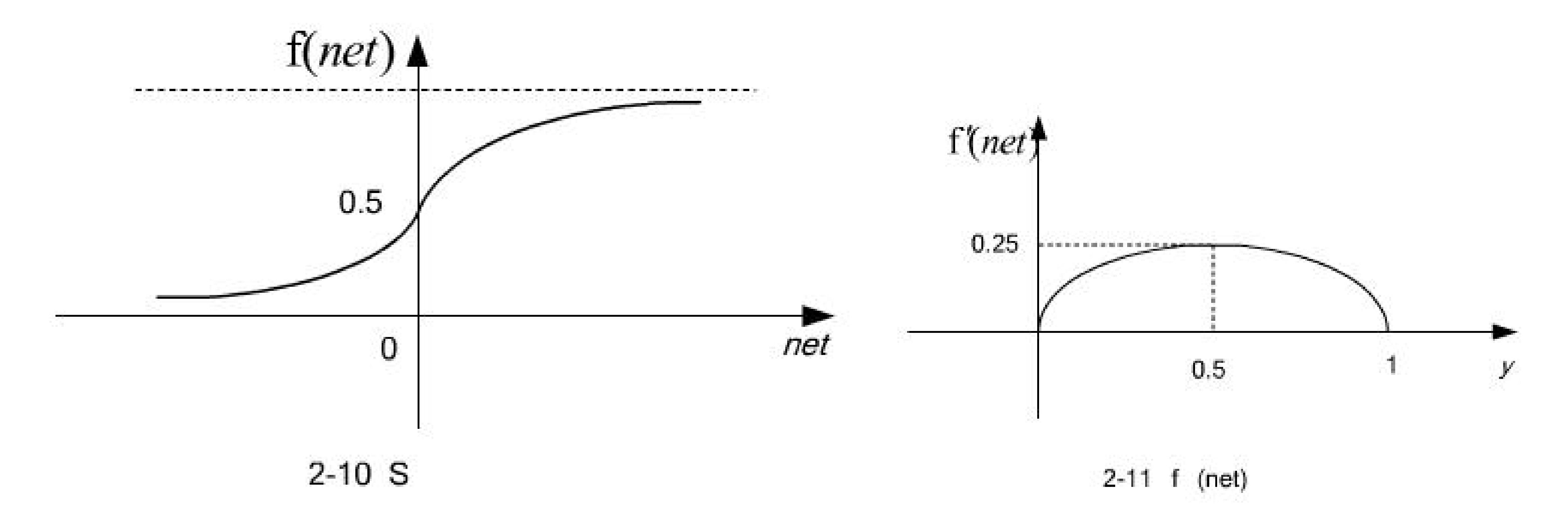
现求 y 关于 net 的导数：

$$f'(net)=\frac{e^{-net}}{(1+e^{-net})^2}=\frac{1+e^{-net}-1}{(1+e^{-net})^2}=\frac{1}{1+e^{-net}}-\frac{1}{(1+e^{-net})^2}=y-y^2=y(1-y)$$

注意到：$\lim\limits_{net\to+\infty}\frac{1}{1+e^{-net}}=1$，$\lim\limits_{net\to-\infty}\frac{1}{(1+e^{-net})^2}=0$

根据 S 型激活函数可知，y 的值域为 (0，1)，从而，$f'(net)$ 的值域为(0，0.25)，而且是在 y 为 0.5 时，$f'(net)$ 达到极大值，如图 2-4-3 所示。

由图 2-4-2、图 2-4-3，可以看出，对神经网络进行训练，应该将 net 的值尽量控制在收敛比较快的范围内。实际上，也可以用其它函数作为 BP 网络神经元的激活函数，只要该函数满足处处可导的条件即可。



2-10 S　　　　2-11 f (net)

2.4.2 BP 网络的标准学习算法

本节以典型的三层 BP 网络为例，描述标准的 BP 算法。标准 BP 算法是基于梯度下降法的学习算法，学习过程是通过调整权值和阈值，使输出期望值和神经网络实际输出值的均方误差趋于最小而实现的，但是它只用到均方误差函数对权值和阈值的一阶导数（梯度）的信息，使得算法存在收敛速度缓慢、易陷入局部极小等缺陷。

为了算法描述的方便，先定义下面向量和变量：

输入向量$\mathbf{x}=(x_1,x_2,\cdots$ ；

隐含层输入向量$\mathbf{hi}=(hi_1,hi_2,\cdots$ ；

隐含层输出向量$\mathbf{ho}=(ho_1,ho_2,\cdots$ ；

输出层输入向量$\mathbf{yi}=(yi_1,yi_2,\cdots$ ；

输出层输出向量$\mathbf{yo}=(yo_1,yo_2,\cdots$ ；

期望输出向量$\mathbf{d}=(d_1,d_2,\cdots$ ；

输入层与中间层的连接权值w_{ih}；

隐含层与输出层的连接权值w_{ho}；

隐含层各神经元的阈值b_h；

输出层各神经元的阈值b_o；

样本数据个数$k=1,2,\cdots$ ；

激活函数$\mathrm{f}(\cdot)$。

BP 标准算法具体实现步骤如下：

1.网络初始化 给w_{ih}、w_{ho}、b_h和b_o分别赋一个区间（-1，1）内的随机数，设定误差函数$e=\frac{1}{2}\sum_{o=1}^{q}(d_o(k)-y_o(k))^2$，给定计算精度值$\varepsilon$和最大学习次数$M$。

2.随机选取第k个输入样本$\mathbf{x}(k)=(x_1(k),x_2(k),\cdots$ 及对应的期望输出$\mathbf{d}(k)=(d_1(k),d_2(k),\cdots$ 。

3.计算隐含层各神经元的输入 $hi_h(k)$，然后用 $hi_h(k)$ 和激活函数计算隐含层各神经元的输出 $ho_h(k)$。

$$hi_h(k)=\sum_{i}^{n} w_{ih}x_i(k)-b_h \qquad h=1,2,\cdots$$

$$ho_h(k)=\mathrm{f}(hi_h(k)) \qquad h=1,2,\cdots$$

$$yi_o(k)=\sum_{h}^{p} w_{ho}ho_h(k)-b_o \qquad o=1,2,\cdots$$

$$yo_o(k)=\mathrm{f}(yi_o(k)) \qquad o=1,2,\cdots$$

4.利用网络期望输出向量 $\mathbf{d}(k)=\left(d_1(k),d_2(k),\cdots\right)$，网络的实际输出 $yo_o(k)$，计算误差函数对输出层的各神经元的偏导数 $\delta_o(k)$。

$$\delta_o(k)=(d_o(k)-yo_o(k))\cdot yo_o(k)(1-yo_o(k)) \qquad o=1,2,\cdots$$

5.利用隐含层到输出层的连接权值 $w_{ho}(k)$、输出层的 $\delta_o(k)$ 和隐含层的输出 $ho_h(k)$ 计算误差函数对隐含层各神经元的偏导数 $\delta_h(k)$。

$$\delta_h(k)=\left[\sum_{o=1}^{q}\delta_o(k)w_{ho}\right]ho_h(k)(1-ho_h(k))$$

6.利用输出层各神经元的 $\delta_o(k)$ 和隐含层各神经元的输出 $ho_h(k)$ 来修正连接权值 $w_{ho}(k)$ 和阈值 $b_o(k)$。

$$w_{ho}^{N+1}(k)=w_{ho}^{N}(k)+\eta\delta_o(k)ho_h(k)$$

$$b_o^{N+1}(k)=b_o^{N}(k)+\eta\delta_o(k)$$

式中，N 表示调整前，$N+1$ 表示调整后，η 为学习率，在（0，1）之间取值。

7.使用隐含层各神经元的 $\delta_h(k)$ 和输入层各神经元的输入 $x_i(k)$ 修正连接权和阈值。

$$w_{ih}^{N+1} = w_{ih}^{N} + \eta\delta_h(k)x_i(k)$$

$$b_h^{N+1}(k) = b_h^{N+1}(k) + \eta\delta_h(k)$$

8.计算全局误差 E

$$E = \frac{1}{2m}\sum_{k=1}^{m}\sum_{o=1}^{q}(d_o(k) - y_o(k))^2$$

9.判断网络误差是否满足要求 当 $E < \varepsilon$ 或学习次数大于设定的最大次数 M，则结束算法。否则，随机选取下一个学习样本及对应的期望输出，返回到第 3 步，进入下一轮学习过程。

实际使用中，还应该对已经训练好的网络进行测试，即用一组与训练样本不完全相同的测试样本数据输入到已经训练好的网络中，计算其得到的结果是否在规定的精度范围内。

2.4.3 BP 神经网络学习算法的 MATLAB 实现

MATLAB 神经网络工具箱提供了的与本算法相关的 BP 网络分析和设计的工具箱函数如表 2-4-1 所示。在 MATLAB 的命令行窗口中输入“help backprop”，便可得到与 BP 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍

表 2-4-1 MATLAB 中 BP 神经网络的重要函数和基本功能

函数名	功能
newff()	生成一个前馈 BP 网络
tansig()	双曲正切 S 型(Tan-Sigmoid)传输函数
logsig()	对数 S 型(Log-Sigmoid)传输函数
traingd()	梯度下降 BP 训练函数

下面将对表 2-4-1 中的工具函数的使用进行说明，并通过一个药品销售预测的实例来说明如何

建立一个解决实际应用问题的 BP 神经网络。

1. Newff()

功能：建立一个前向 BP 网络

格式：(1)　net = newff(PR，[S1 S2...SN1]，{TF1 TF2...TFN1}，BTF，BLF，PF)

说明：　net 为创建的新 BP 神经网络；PR 为网络输入取向量取值范围的矩阵；[S1 S2…SNl]表示网络隐含层和输出层神经元的个数；{TFl TF2…TFN1}表示网络隐含层和输出层的传输函数，默认为‘tansig’；BTF 表示网络的训练函数，默认为‘trainlm’；BLF 表示网络的权值学习函数，默认为‘learngdm’；PF 表示性能数，默认为‘mse’。该函数可以建立一个 N 层前向 BP 网络。各神经元权值和阈值的初始化函数为 initnw，网络的自适应调整函数为 trains，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。

2. Tansig()

功能：正切 sigmoid 激活函数

格式：(1) a = tansig(n)

双曲正切 Sigmoid 函数把神经元的输入范围从(-∞，+∞)映射到(-1，1)。它是可导函数，适用于 BP 训练的神经元。

3.　Logsig()

功能：对数 Sigmoid 激活函数

格式：(1) a = logsig(N)

说明：函数 logsig()的用法同上。如果 BP 网络的最后一层是 Sigmoid 型神经元，那么整个网络的输出就被限制在一个较小的范围内；如果 BP 网络的最后一层是 Purelin 型线性神经元，那么整个网络的输出可以取任意值。

例子 2-3　表 2-4-2 为某药品的销售情况，现构建一个如下的三层 BP 神经网络对药品的销售进行预测：输入层有三个结点，隐含层节点数为 5，隐含层的激活函数为 tansig(双曲正切 S 型传递函数)；输出层结点数为 1 个，输出层的激活函数为 logsig(S 型的对数函数)，并利用此网络对药品的销售量进行预测，预测方法采用滚动预测方式，即用前三个月的销售量来预测第四个月的销售量，如用 1、2、3 月的销售量为输入预测第 4 个月的销售量，用 2、3、4 月的销售量为输入预测第 5 个月的销售量.如此反复直至满足预测精度要求为止。

表 2-4-2 药品销售情况表

月份	1	2	3	4	5	6
销量	2056	2395	2600	2298	1634	1600
月份	7	8	9	10	11	12
销量	1873	1478	1900	1500	2046	1556

```
%以每三个月的销售量经归一化处理后作为输入
P=[0.5152    0.8173    1.0000 ;
    0.8173    1.0000    0.7308;
    1.0000    0.7308    0.1390;
    0.7308    0.1390    0.1087;
    0.1390    0.1087    0.3520;
    0.1087    0.3520    0.0000;]';
%以第四个月的销售量归一化处理后作为目标向量
T=[0.7308 0.1390 0.1087 0.3520 0.0000 0.3761];
%创建一个 BP 神经网络，每一个输入向量的取值范围为[0 ,1]，隐含层有 5 个神经元，输出层
%有一个神经元，隐含层的激活函数为 tansig，输出层的激活函数为 logsig，训练函数为梯度下
%降函数，即 2.3.2 节中所描述的标准学习算法
net=newff([0 1;0 1;0 1],[5,1],{'tansig','logsig'},'traingd');
net.trainParam.epochs=15000;
net.trainParam.goal=0.01;
%设置学习速率为 0.1
LP.lr=0.1;
```

net=train(net,P,T);

训练结束后，运行后得到图 2-4-4 在命令行窗口中的输出结果如下

……

TRAINGD, Epoch 12650/15000, MSE 0.0100199/0.01, Gradient 0.00807289/1e-010

TRAINGD, Epoch 12675/15000, MSE 0.0100036/0.01, Gradient 0.00806391/1e-010

TRAINGD, Epoch 12681/15000, MSE 0.00999968/0.01, Gradient 0.00806176/1e-010

TRAINGD, Performance goal met.

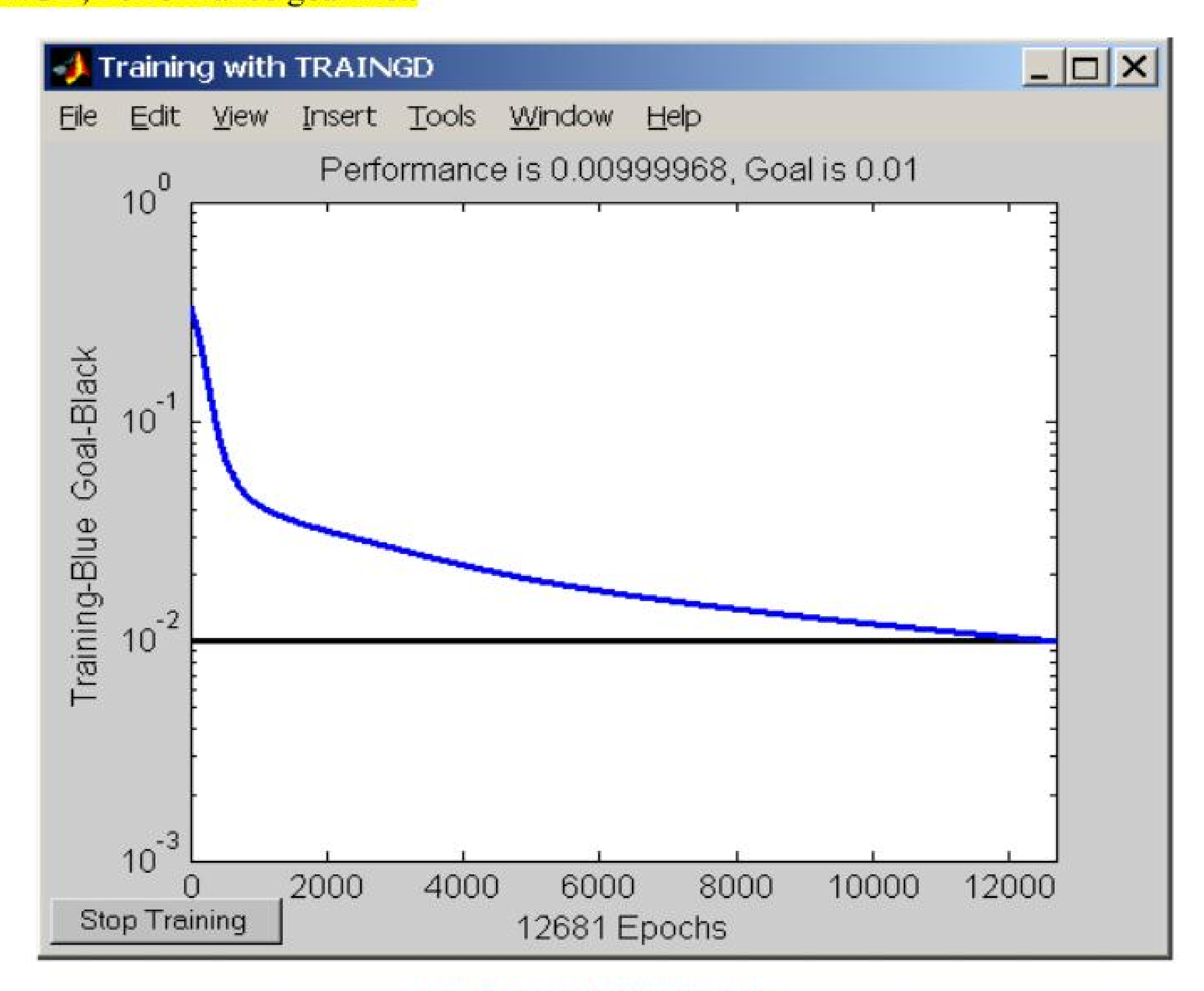


图 2-4-4 训练误差曲线

由图 2-4-4 和输出结果可以得出训练过程在 10511 次训练时结束并达到预定的目标误差要求。

2.5 径向基函数神经网络模型与学习算法

1985 年，Powell 提出了多变量插值的径向基函数(Radical Basis Function，RBF)方法。1988 年，Moody 和 Darken[10,11]提出了一种神经网络结构，即 RBF 神经网络，属于前向神经网络类型，它能够以任意精度逼近任意连续函数，特别适合于解决分类问题。

RBF 网络的结构与多层前向网络类似，它是一种三层前向网络。输入层由信号源结点组成；第二层为隐含层，隐单元数视所描述问题的需要而定，隐单元的变换函数是 RBF，它是对中心点径向对称且衰减的非负非线性函数；第三层为输出层，它对输人模式的作用作出响应。从输人空间到隐含层空间的变换是非线性的，而从隐含层空间到输出层空间变换是线性的。

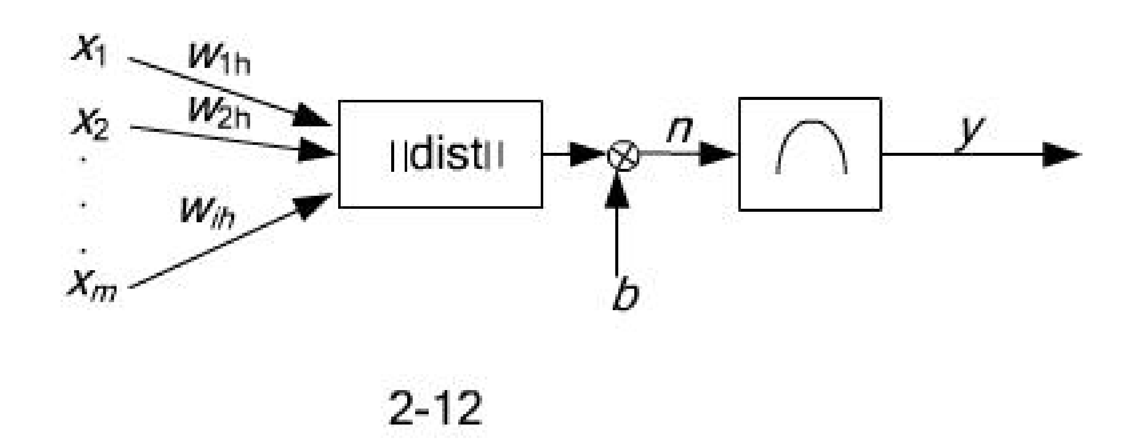
RBF 网络的基本思想是：用 RBF 作为隐单元的“基”构成隐含层空间，这样就可将输入矢量直接(即不需要通过权连接)映射到隐空间。当 RBF 的中心点确定以后，这种映射关系也就确定了。而隐含层空间到输出空间的映射是线性的，即网络的输出是隐单元输出的线性加权和，此处的权即为网络可调参数。由此可见，从总体上看，网络由输人到输出的映射是非线性的，而网络输出对可调参数而言却又是线性的。这样网络的权就可由线性方程组直接解出，从而大大加快学习速度并避免局部极小问题。

2.5.1RBF 神经网络模型

径向基神经网络的神经元结构如图 2-5-1 所示。径向基神经网络的激活函数采用径向基函数，通常定义为空间任一点到某一中心之间欧氏距离的单调函数。由图 2-12 所示的径向基神经元结构可以看出，径向基神经网络的激活函数是以输入向量和权值向量之间的距离 $\|dist\|$ 作为自变量的。径向基神经网络的激活函数的一般表达式为：

$$R(\|dist\|)=e^{-\|dist\|^2}$$

随着权值和输入向量之间距离的减少，网络输出是递增的，当输入向量和权值向量一致时，神经元输出 1。结构中的 b 为阈值，用于调整神经元的灵敏度。利用径向基神经元和线性神经元可以建立广义回归神经网络，该种神经网络适用于函数逼近方面的应用；径向基神经元和竞争神经元（2.5 节介绍）可以组建概率神经网络，此种神经网络适用于解决分类问题。



2-12

由输入层、隐含层和输出层构成的一般径向基神经网络结构如图 2-5-2 所示。在 RBF 网络中，输入层仅仅起到传输信号的作用，与前面所讲述的神经网络相比较，输入层和隐含层之间可以看作连接权值为 1 的连接。输出层和隐含层所完成的任务是不同的，因而它们的学习策略也不相同。输出层是对线性权进行调整，采用的是线性优化策略，因而学习速度较快。而隐含层是对激活函数(格林函数或高斯函数，一般取高斯函数)的参数进行调整，采用的是非线性优化策略，因而学习速度较慢。

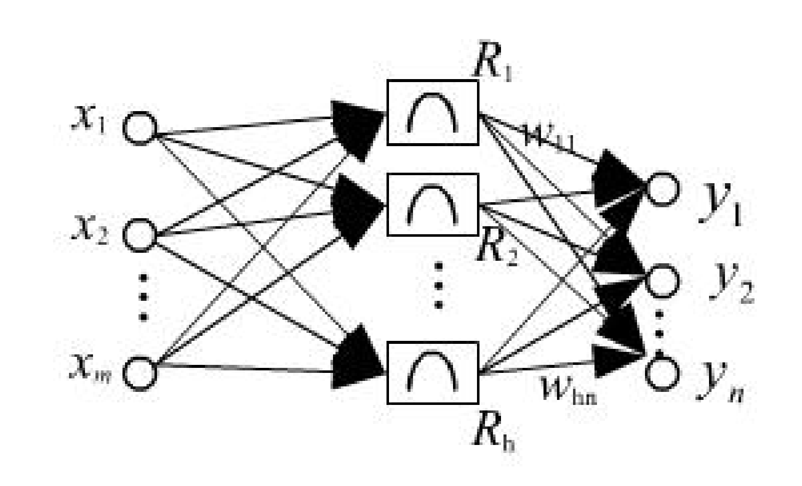


图2-13 径向基神经网络结构

尽管 RBF 网络的输出是隐单元输出的线性加权和，学习速度加快，但并不等于径向基神经网络就可以取代其它前馈网络。这是因为径向基神经网络很可能需要比 BP 网络多得多的隐含层神经网络元来完成工作。BP 网络使用 sigmoid()函数，这样的神经元有很大的输入可见区域，而径向基神经网络使用的径向基函数，输入空间区域很小。这就不可避免地导致了在输入空间较大时，需要更多的径向基神经元。

2.5.2RBF 网络的学习算法

RBF 神经网络学习算法需要求解的参数有 3 个：基函数的中心、方差以及隐含层到输出层的权值。根据径向基函数中心选取方法的不同，RBF 网络有多种学习方法，如随机选取中心法、自组织

选取中心法、有监督选取中心法和正交最小二乘法等。下面将介绍自组织选取中心的 RBF 神经网络学习法。此方法由两个阶段组成：一是自组织学习阶段：此阶段为无导师学习过程，求解隐含层基函数的中心与方差；二是有导师学习阶段：此阶段求解隐含层到输出层之间的权值。

径向基神经网络中常用的径向基函数是高斯函数，因此径向基神经网络的激活函数可表示为：

$$\mathrm{R}(\mathbf{x}_p - c_i)=\exp(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_p - c_i\right\|^2)$$

式中，$\|\cdot\|$ 表示欧氏范数，c 表示高斯函数的中心，σ 表示高斯函数的方差。

由图 2-5-2 的径向基神经网络的结构可得到网络的输出为：

$$\mathrm{y}_j=\sum_{\mathrm{i=1}}^{\mathrm{h}} w_{ij}\exp(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_p - c_i\right\|^2) \qquad j=1,2,\cdots$$

式中，$\mathbf{x}_p=(x_1^P,x_2^P,\cdots$ 是第 p 个输入样本，$p=1,2,\cdots$ ，P 表示样本总数；c_i 表示网络隐含层结点的中心，w_{ij} 表示隐含层到输出层的连接权值，$i=1,2,\cdots$ ，h 表示隐含层的节点数，y_j 表示与输入样本对应的网络的第 j 个输出节点的实际输出。设 d 是样本的期望输出值，那么基函数的方差可表示为：

$$\sigma=\frac{1}{P}\sum_{j}^{m}\left\|d_j - y_j c_i\right\|^2$$

学习算法具体步骤如下：

1.基于 K-均值聚类方法求取基函数中心 c

（1） 网络初始化 随机选取 h 个训练样本作为聚类中心 c_i（$i=1,2,\cdots$ ）。

（2）将输入的训练样本集合按最近邻规则分组 按照 x_p 与中心为 c_i 之间的欧氏距离将 x_p 分配到输入样本的各个聚类集合 $\vartheta_p(p=1,2,\cdots$ 中。

（3）重新调整聚类中心 计算各个聚类集合 ϑ_p 中训练样本的平均值，即新的聚类中心 c_i，如果新的聚类中心不再发生变化，则所得到的 c_i 即为 RBF 神经网络最终的基函数中心，否则返回（2），

进入下一轮的中心求解。

2.求解方差 σ_i 该 RBF 神经网络的基函数为高斯函数，因此方差 σ_i 可由下式求解：

$$\sigma_i = \frac{c_{\max}}{\sqrt{2h}}, i = 1, 2, \cdots$$

式中，$c_{\max}$ 为 1 中所选取中心之间的最大距离。

3.计算隐含层和输出层之间的权值 隐含层至输出层之间神经元的连接权值可以用最小二乘法直接计算得到，计算公式如下：

$$w = \exp\left(\frac{h}{c_{\max}^2}\left\|x_p - c_i\right\|^2\right) \qquad p = 1, 2, \cdots \qquad \cdots$$

2.5.3RBF 网络学习算法的 MATLAB 实现

MATLAB 神经网络工具箱提供的与算法相关的径向基神经网络的工具函数如表 2-5-1 所示。在 MATLAB 的命令行窗口中输入“help radbasis”，便可得到与径向基神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。

表 2-5-1 径向基网络的重要函数和基本功能

函数名	功能
newrb()	新建一个径向基神经网络
newrbe()	新建一个严格的径向基神经网络
newgrnn()	新建一个广义回归径向基神经网络
newpnn()	新建一个概率径向基神经网络

下面将对表 2-5-1 中的工具函数的使用进行说明，并通过一个实例来说明如何使用表中的工具函数建立一个线性神经网络。

1、Newrb()

功能：建立一个径向基神经网络

格式：(1) net = newrb(P，T，GOAL，SPREAD，MN，DF)

说明：P 为输入向量，T 为目标向量，GOAL 为均方误差，默认为 0，SPREAD 为径向基函数

的分布密度，默认为 1，MN 为神经元的最大数目，DF 为两次显示之间所添加的神经元神经元数目。

2、Newrbe()

功能：建立一个严格的径向基神经网络

格式：(1)　net = newrb(P，T，　SPREAD)

说明：各参数的含义见 Newrb。

3、Newgrnn()

功能：建立一个广义回归径向基神经网络

格式：(1)　net = newrb(P，T，　SPREAD)

说明：各参数的含义见 Newrb。

4、Newgpnn()

功能：建立一个概率径向基神经网络

格式：(1)　net = newrb(P，T，　SPREAD)

说明：各参数的含义见 Newrb。

例子 2-4　建立一个径向基神经网络，对非线性函数 y=sqrt(x)进行逼近，并作出网络的逼近误差曲线。

```
%输入从 0 开始变化到 5，每次变化幅度为 0.1
x=0:0.1:5;
y=sqrt(x);
%建立一个目标误差为 0，径向基函数的分布密度为 0.5，隐含层神经元个数的最大值为 20，每
%增加 5 个神经元显示一次结果
net=newrb(x,y,0,0.5,20,5);
t=sim(net,x);
%在以输入 x 和函数值与网络输出之间的差值 y-t 坐标上绘出误差曲线，并用“*”来标记函数
%值与网络输出之间的差值
plot(x,y-t,'*-')
```

训练结束后，网络的逼近误差曲线如图 2-5-3 所示，在命令行窗口中的输出结果如下：

```
%用 NEWRB 来训练网络，神经元个数为 0，均方和误差为 9.17903
NEWRB, neurons = 0, SSE = 9.17903
NEWRB, neurons = 5, SSE = 0.924825
NEWRB, neurons = 10, SSE = 0.0394536
NEWRB, neurons = 15, SSE = 0.00184102
NEWRB, neurons = 20, SSE = 3.75484e-005
```

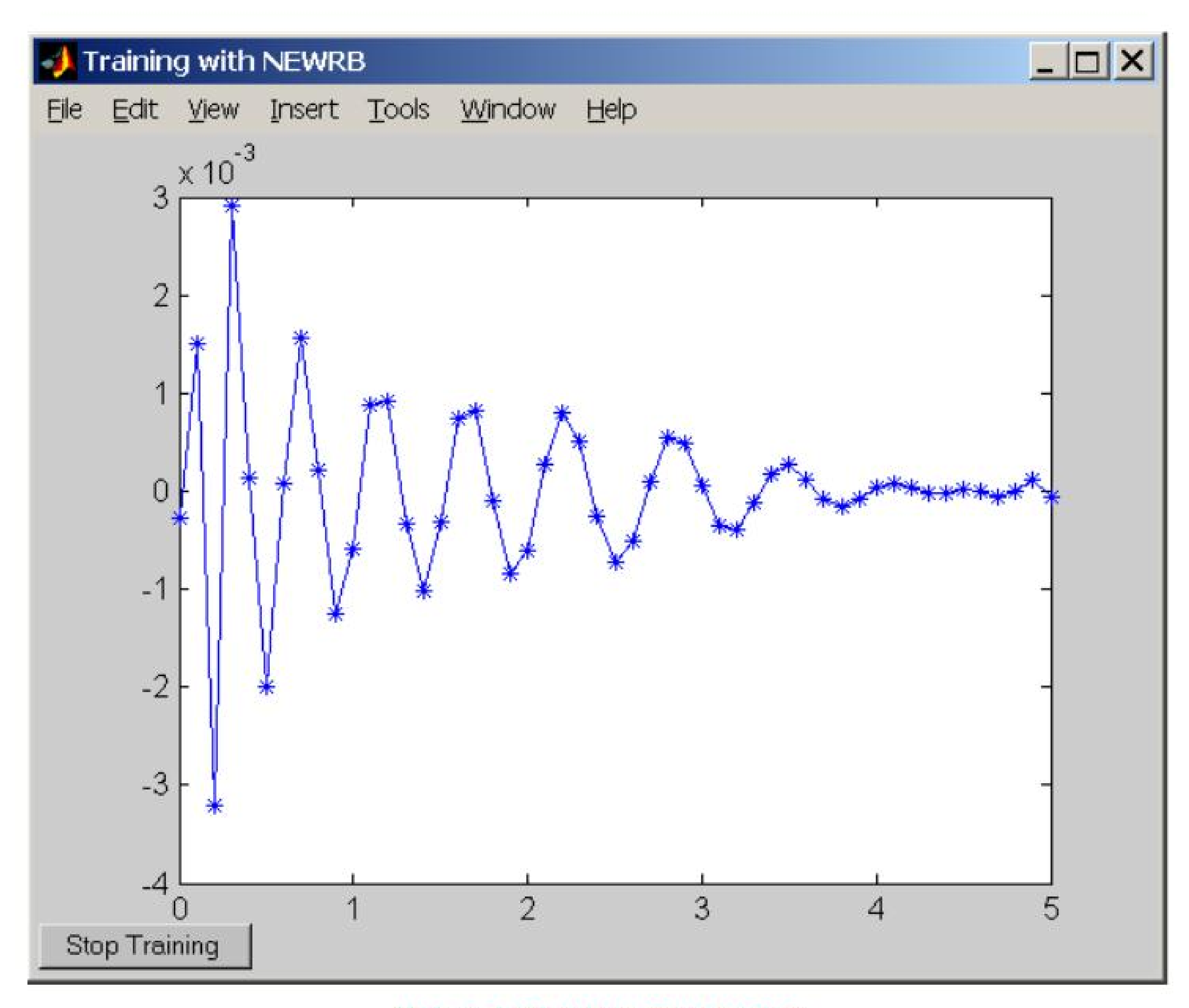


图 2-5-3 网络的逼近误差曲线

由图 2-5-3 可以看到，网络的输出和函数值之间的差值在隐含层神经元的个数为 20 时已经接近于 0，说明网络输出能非常好地逼近函数。

2.6 自组织神经网络模型与学习算法

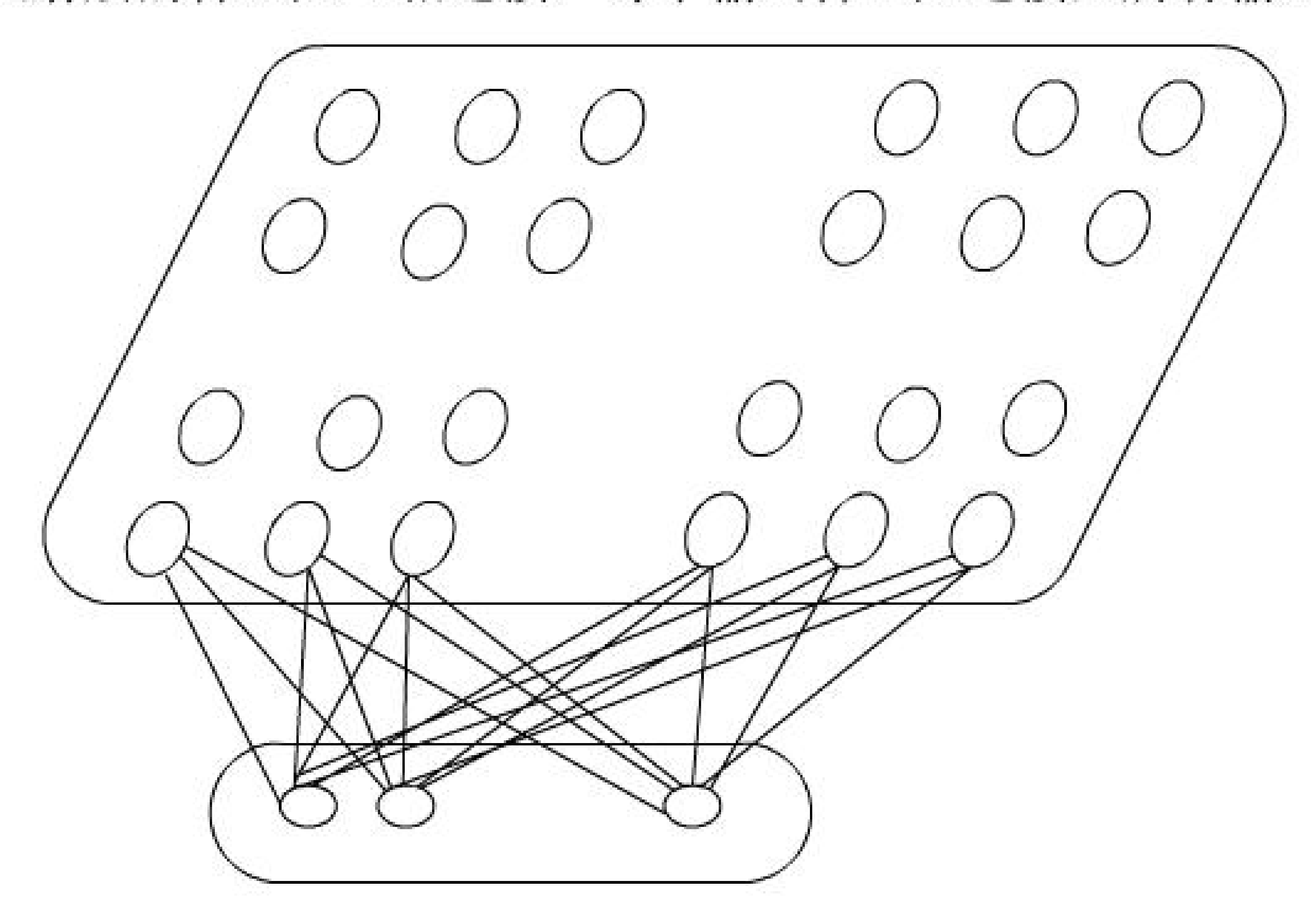
自组织神经网络，又称为自组织竞争神经网络，特别适合于解决模式分类和识别方面的应用问题。自组织神经网络属于前向神经网络类型，采用无导师学习算法，其工作的基本思想是让竞争层各神经元通过竞争与输入模式进行匹配，最后仅有一个神经元成为竞争的胜者，这一获胜神经元的输出就代表对输入模式的分类。自组织特征映射神经网络不仅能够像自组织竞争神经网络一样学习输入的分布情况，而且可以学习神经网络的拓扑结构。

常用的自组织竞争神经网络有自适应共振理论(Adaptive Resonance Theory，ART)网络、自组织特征映射(self-Organizing Feature Map，SOM)网络、对传(Counter Propagation，CP)网络和协同神经网络(Synergetic Neural Network．SNN)等。

本节以及下一节将分别介绍自组织特征映射网络、学习矢量量化两种自组织竞争神经网络的网络模型、学习算法及对应的 MATLAB 工具。

2.6.1 自组织特征映射神经网络结构

自组织特征映射网络也称为 Kohonen 网络，其网络结构如图 2-6-1 所示，它是由荷兰学者 Teuvo Kohonen 于 1981 年提出来的，基本上为输入层和映射层的双层结构，输入层用于接收输入模式，映射层输出结果，映射层的神经元互相连接，每个输出神经元连接至所有输入神经元。



2-14 Kohonen

2.5.2 自组织特征映射网络的学习算法

下面讨论作为无导师学习算法代表的 Kohonen 自组织特征映射算法。Kohonen 自组织特征映射算法，能够自动找出输入数据之间的类似度，将相似的输入在网络上就近配置。因此是一种可以构成对输入数据有选择地给予反应的网络。

Kohonen 的自组织特征映射的学习算法步骤归纳如下：

1.网络初始化 用随机数设定输入层和映射层之间权值的初始值。

2.输入向量的输入 把输入向量 $\mathbf{x}=\left[x_1,x_2,x_3,\cdots\right]^T$ 输入给输入层。

3.计算映射层的权值向量和输入向量的距离 在映射层，计算各神经元的权值向量和输入向量的欧氏距离。这里，映射层的第 j 个神经元和输入向量的距离，按下式给出：

$$d_j = \sqrt{\sum_{i=1}^{n}(x_i - w_{ij})^2}$$

式中，w_{ij} 为输入层的 i 神经元和映射层的 j 神经元之间的权值。

4.选择与权值向量的距离最小的神经元 计算并选择使输入向量和权值向量的距离最小的神经元，如 d_j 为最小，把其称为胜出神经元并记为 j^*，并给出其邻接神经元集合。

5.权值的学习 胜出神经元和位于其邻接神经元的权值，按下式更新：

$$\Delta w_{ij} = \eta \mathrm{h}(j, j^*)(x_i - w_{ij})$$

式中，η 为一个大于 0 小于 1 的常数，$\mathrm{h}(j, j*)$ 成为邻域函数，用下式表示：

$$\mathrm{h}(j, j*) = \exp(-\frac{\left|j - j^*\right|^2}{\sigma^2}) \qquad (2\text{-}3)$$

上式的 σ^2 随着学习的进行而减小。因此，$\mathrm{h}(j, j*)$ 的范围，学习初期很宽，随着学习的进行而变窄。也就是说，随着学习的进行从粗调整向微调整变化。这样，邻域函数 $\mathrm{h}(j, j*)$ 可以起到产生有效映射的作用。

6.是否达到预先设定的要求，如达到要求则算法结束，否则返回 2，进入下一轮学习。

在自组织特征映射中，由式(2-3)可见，胜出神经元和其附近的神经元全部接近当时的输入向量。学习初期，根据邻域函数 $\mathrm{h}(j, j*)$，在附近有很多神经元，形成粗略的映射。随着学习的进行，$\mathrm{h}(j, j*)$ 变窄，胜出神经元附近的神经元数变少。因此，接着继续进行局部微调整，空间分辨率提高。

自组织竞争神经网络算法能够进行有效的自适应分类，但它仍存在一些问题。第一个问题就是学习速度的选择使其不得不在学习速度和最终权值向量的稳定性之间进行折衷。第二个问题是有时一个神经元的初始权值向量离输入向量太远以至于它从未在竞争中获胜，因而也从未得到学习，这将形成毫无用处的“死”神经元。

2.6.3 自组织网络学习算法的 MATLAB 实现

MATLAB 神经网络工具箱提供的与算法相关的自组织神经网络工具函数的名称和基本功能如表 2-6-1 所示。在 MATLAB 的命令行窗口中输入“help selforg”，便可得到与自组织网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。

表 2-6-1 自组织神经网络的重要函数和基本功能

函数名	功能
newsom()	创建一个自组织特征映射神经网络
plotsom()	绘制自组织特征映射网络的权值矢量
vec2ind()	将单值矢量组变换成下标矢量
compet()	竞争传输函数
midpoint()	中点权值初始化函数
learnsom()	自组织特征映射权值学习规则函数

下面对表 2-6-1 中的工具函数的使用进行说明，并通过一个实例来说明如何使用表中的工具函数建立一个自组织神经网络，解决实际应用中的问题。

1. Newsom()

功能：创建一个自组织特征映射网络函数

格式：(1) net = newsom

(2) net = newsom(PR，[D1，D2，...]，TFCN，DFCN，OLR，OSTEPS，TLR，TND)

说明：式(1)返回一个没有定义结构的空对象，并显示图形解雇界面函数 nntool 的帮助文字；式(2)中 net 为生成的新 BP 神经网络；PR 为网络输入矢量取值范围的矩阵[Pmin Pmax]；[D1，D2，...]为神经元在多维空间中排列时各维的个数；TFCN 为拓扑函数，缺省值为 hextop；DFCN 为距离函数，缺省值为 linkdist；OLR 为排列阶段学习速率，缺省值为 0.9；OSTEPS 为排列阶段学习次数，缺省值为 1000；TLR 为调整阶段学习速率，缺省值为 0.02，TND 为调整阶段领域半径，缺省值为 1。自组织映射网络常用于解决分类问题，神经元的加权函数为 netdist，输入函数为 netsum，传递函数为 compet，权值初始化函数为 midpoint，自适应调整函数为 trains，训练函数为 trainr，学习函数为 learnsom。

2. Plotsom()

功能：绘制自组织特征映射网络图的权值向量函数

格式：(1) plotsom(pos)

(2) plotsom(W，D，ND)

说明：式中 pos 是网络中各神经元在物理空间分布的位置坐标矩阵；函数返回神经元物理分布的拓扑图，图中每两个间距小于 1 的神经元以直线连接；W 为神经元权值矩阵；D 为根据神经元位置计算出的间接矩阵；ND 为领域半径，缺省值为 1；函数返回神经元权值的分布图，图中每两个间距小于 ND 的神经元以直线连接。

3. Vec2ind()

功能：将单值向量组变换成下标向量

格式：ind = vec2ind(vec)

说明：式中，vec 为 m 行 n 列的向量矩阵 x，x 中的每个列向量 i，除包含一个 1 外，其余元素均为 0， ind 为 n 个元素值为 1 所在的行下标值构成的一个行向量。

例子 2-5 人口分类是人口统计中的一个重要指标，现有 1999 共 10 个地区的人口出生比例情况如下：

出生男性百分比分别为：0.5512 0.5123 0.5087 0.5001 0.6012 0.5298 0.5000 0.4965 0.5103 0.5003;

出生女性百分比分别为：0.4488 0.4877 0.4913 0.4999 0.3988 0.4702 0.5000 0.5035 0.4897 0.4997

建立一个自组织神经网络对上述数据分类，给定某个地区的男、女出生比例分别为 0.5，0.5，测试训练后的自组织神经网络的性能，判断其属于哪个类别。

MATLAB 代码如下：

```
P=[0.5512   0.5123   0.5087   0.5001   0.6012   0.5298   0.5000   0.4965   0.5103   0.5003;
0.4488   0.4877   0.4913   0.4999   0.3988   0.4702   0.5000   0.5035   0.4897   0.4997];
%创建一个自组织神经网络，[0 1;0 1]表示输入数据的取值范围在[0，1]之间，[3，4]表示竞争
%层组织结构为 3×4，其余参数取默认值
net=newsom([0 1;0 1],[3 4]);
net.trainParam.epochs=500;
net=init(net);
net=train(net,P);
y=sim(net,P);
%获取训练后的自组织神经网络的权值
w1=net.IW{1,1};
%绘出训练后自组织神经网络的权值分布图
plotsom(w1,net.layers{1}.distances);
%输入测试数据
```

```
p=[0.5;0.5];
%对网络进行测试
y_test=sim(net,p);
%将测试数据所得到的将单值向量组变换成下标向量
y_test=vec2ind(y_test)
```

程序运行后得到训练后的权值分布如图 2-6-2 所示，在命令行窗口得到下面结果：

```
%使用 TRAINR 作为训练函数，最大训练次数为 500 次
>> TRAINR, Epoch 0/500
TRAINR, Epoch 25/500
TRAINR, Epoch 50/500
TRAINR, Epoch 75/500
TRAINR, Epoch 100/500
TRAINR, Epoch 125/500
TRAINR, Epoch 150/500
TRAINR, Epoch 175/500
TRAINR, Epoch 200/500
TRAINR, Epoch 225/500
TRAINR, Epoch 250/500
TRAINR, Epoch 275/500
TRAINR, Epoch 300/500
TRAINR, Epoch 325/500
TRAINR, Epoch 350/500
TRAINR, Epoch 375/500
TRAINR, Epoch 400/500
TRAINR, Epoch 425/500
TRAINR, Epoch 450/500
TRAINR, Epoch 475/500
TRAINR, Epoch 500/500
TRAINR, Maximum epoch reached.
%训练结束后的结果
y =
     4    10    10    12     1     6    12    12    10    12
%输入测试数据后所得到的结果
y_test =
    12
```

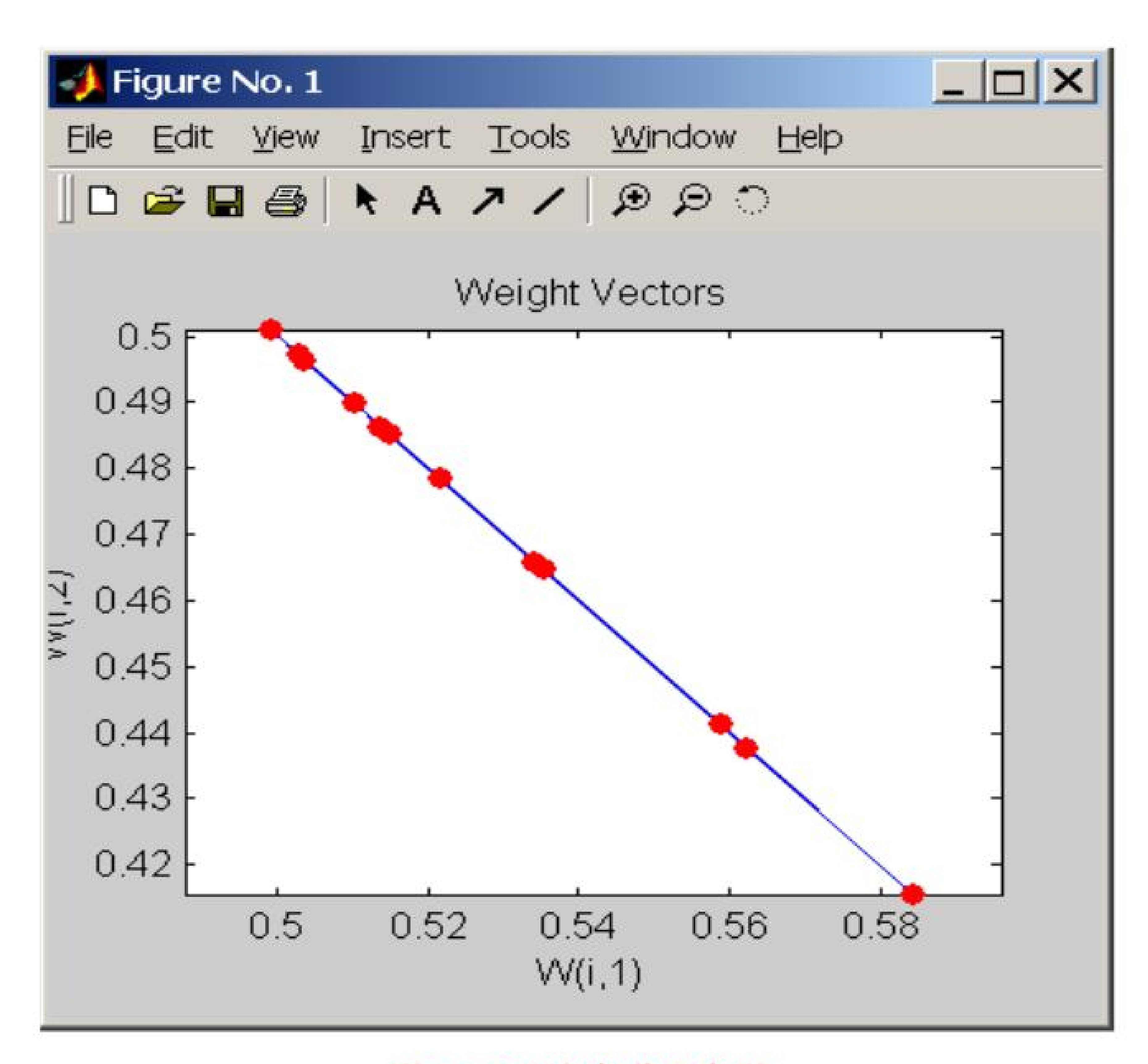


图 2-6-2 网络权值分布图

将训练后的输出数据列到表 2-6-2 中，由表可知网络将数据聚成了 5 类，测试数据被划分到第 3 中。

表 2-6-2 输出结果类型

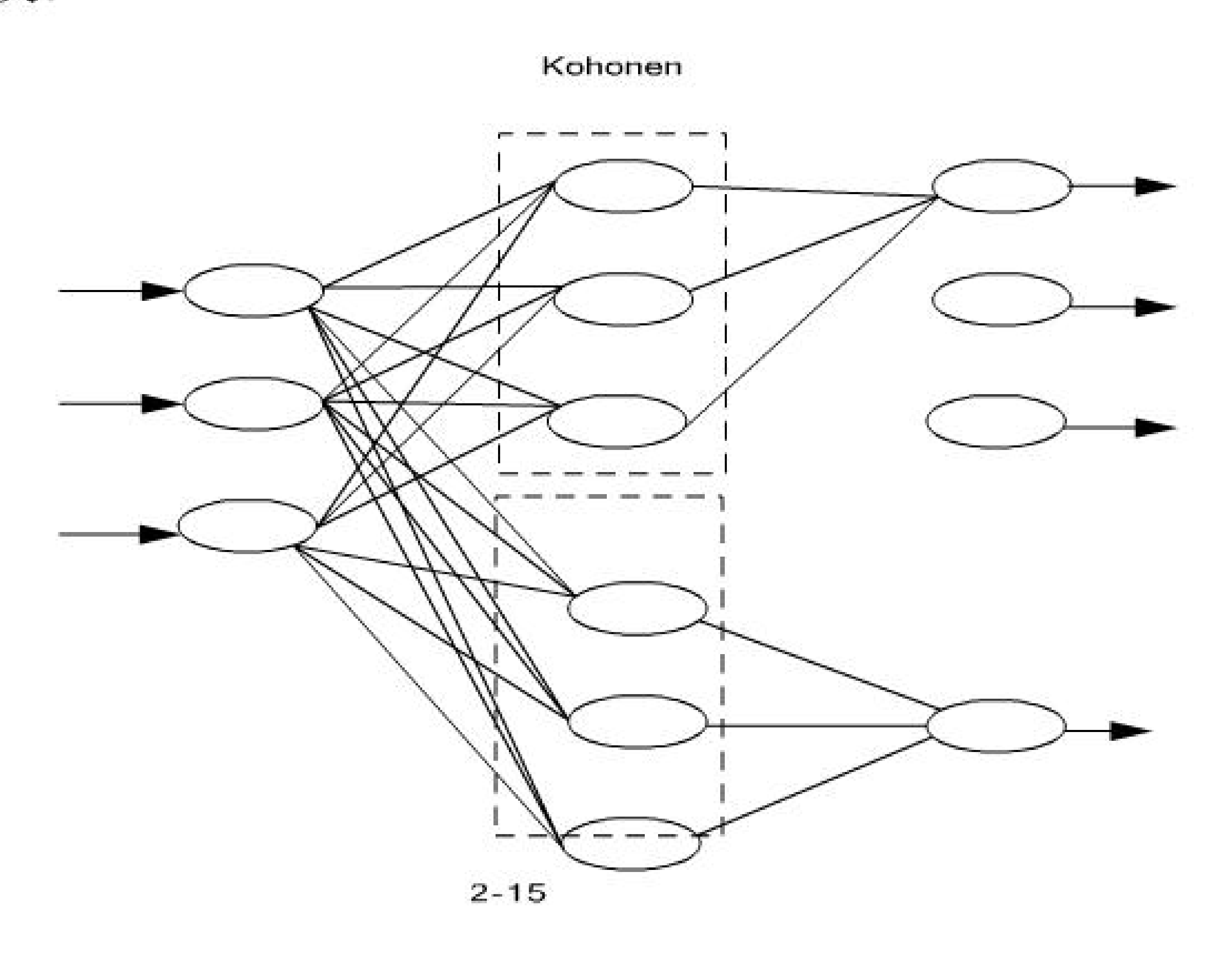
样本数据	类别	激发神经元标号
1	1	4
2，3，9	2	10
4，7，8	3	12
5	4	1
6	5	6

2.7 学习向量量化(LVQ)神经网络模型与学习算法

2.7.1 LVQ 神经网络结构

学习向量量化 LVQ(Learning Vector Quantization)神经网络，属于前向神经网络类型，在模式识别和优化领域有着广泛的的应用，其网络结构如图 2-7-1 所示。

LVQ 神经网络由三层组成，即输入层、隐含层和输出层，网络在输入层与隐含层间为完全连接，而在隐含层与输出层间为部分连接，每个输出层神经元与隐含层神经元的不同组相连接。隐含层和输出层神经元之间的连接权值固定为 1 。输入层和隐含层神经元间连接的权值建立参考矢量的分量（对每个隐含神经元指定一个参考矢量）。在网络训练过程中，这些权值被修改。隐含层神经元（又称为 Kohnen 神经元）和输出神经元都具有二进制输出值。当某个输入模式被送至网络时，参考矢量最接近输入模式的隐含神经元因获得激发而赢得竞争，因而允许它产生一个“1”，而其它隐含层神经元都被迫产生“0”。与包含获胜神经元的隐含层神经元组相连接的输出神经元也发出“1”，而其它输出神经元均发出“0”。产生“1”的输出神经元给出输入模式的类，由此可见，每个输出神经元被用于表示不同的类。



2.7.2 LVQ 神经网络的学习算法

LVQ 神经网络算法是在有教师状态下对隐含层进行训练的一种学习算法，是从 Kohonen 竞争算

法演化而来的。向量量化的想法是将几个数据归到一起，进行模式分类，因此 LVQ 算法可以认为是把自组织特征映射算法改良成有教师学习的算法。LVQ 神经网络算法可分为 LVQ1 算法和 LVQ2 算法两种。

一、LVQ1 算法

LVQ 网络与自组织特征映射网络相同，为双层网络结构。因为 LVQ 网络是以模式分类为目的，一般第 2 层比第 1 层的神经元数少，同一层的神经元之间没有结合。 在 LVQ 中，第 2 层各个神经元有各自的领域。根据第 1 层和第 2 层之间的权值，输入向量可以被分到任意一个领域。Kohonen 把自组织特征映射算法改良成有教师学习算法，首先设计了 LVQl 算法。

LVQ1 算法具体步骤如下：

1.网络初始化 用较小的随机数设定输人层和隐含层之间的权值初始值。

2.输入向量的输入 将输人向量 $\mathbf{x}=[x_1,x_2,x_3,\cdots]^T$ 送入到输入层。

3.计算输出层与输入向量的距离 输出层第 j 个神经元和输入向量的距离，与自组织化映射的情况相同，由下式给出：

$$d_j=\sqrt{\sum_{i=1}^{n}(x_i-w_{ij})^2} \qquad (2\text{-}4)$$

式中，w_{ij} 为输入层的 i 神经元相映射层的 j 神经元之间的权值。

4.选择与权值向量的距离最小的神经元 计算并选择使输入向量和权值向量的距离最小的神经元就，如 d_j 为最小，则把其称为胜出神经元并记为 j^*。

5.更新连接权值 更新连接权值时，正确分类时和不正确分类时的权值的调整更新公式是不同的，公式如下：

$$\Delta w_{ji}=\begin{cases}+\eta(x_i-w_{ji}) & (2\text{-}5)\\ -\eta(x_i-w_{ji}) & (2\text{-}6)\end{cases}$$

式中，$\eta>0$ 为学习系数，如果是正确分类时，权值调整量采用（2-5）式，否则权值调整量采用（2-6）式。

6.是否满足预先设定的精度要求，满足时算法结束，否则返回 2，进入下一轮学习。

二、LVQ2 算法

当 LVQ1 出现识别错误时，即当不应该胜出的神经元以微小的差别成为胜出神经元时，用 LVQ2 对其加以改良。这相当于输入向量恰好临近到两个神经元的领域的交界附近，以微小的差别进入到

错误一方。

具体算法中，1~4 与 LVQl 算法相同。

5.更新连接权值。网络在正确识别时与 LVQ1 的情况相同，根据式（2-5）进行权值的更新。当不应该胜出的神经元 1 以微小的差别成为胜出神经元 2，且满足以下条件时时：

（1）神经元 2 是正确的；

（2）神经元 2 与胜出神经元的差很小。

对胜出神经元 1 和因微小差别而没有胜出的神经元 2，根据式（2-6）进行权值的更新。

6、是否满足预先设定的精度要求，满足时算法结束，否则返回 2，进入下一轮学习。

2.7.3 LVQ 神经网络学习算法的 MATLAB 实现

MATLAB 神经网络工具箱中提供的与算法相关的 LVQ 神经网络工具函数和基本功能如表 2-7-1 所示。在 MATLAB 的命令行窗口中输入“help lvq”，便可得到与 LVQ 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍

表 2-7-1 LVQ 的重要函数和基本功能

函数名	功　能
newlvq()	建立一个 LVQ 神经网络函数
learnlv1()	LVQ1 权值学习函数
vec2ind()	将单值矢量组变换成下标矢量
plotvec()	用不同的颜色画矢量函数

下面对表 2-7-1 中的工具函数的使用进行说明，并通过一个实例来说明如何使用表中的工具函数建立一个 LVQ 神经网络并用于解决应用问题。

1. Newlvq()

功能：建立一个向量量化神经网络函数

格式：(1) net = newlvq

(2) net = newlvq(PR，S1，PC，LR，LF)

说明：式(1)返回一个没有定义结构的空对象，并显示函数 nntool 的帮助文字；式(2)中，net 为生成的学习向量量化网络；PR 为一个 Rx2 维的网络输入矢量取值范围的矩阵[Pmin Pmax]；Sl 表示隐含层神经元的数目；PC 表示在第二层的权值中列所属类别的百分比；LR 表示学习速率，默认值为 0.01；Lf 表示学习函数，默认值为 learnlv1。

2. Ind2vec()

功能：将下标矢量变换成单值矢量组函数

格式：vec = ind2vec(ind)

说明：式中，ind 为包含 n 个下标的行矢量 x；vec 为 m 行 n 列的矢量组矩阵，矩阵中的每个矢量 i，除了由 x 中的第 i 个元素指定的位置为 1 外，其余元素均为 0，矩阵的行数 m 等于 x 中最大的下标值。

3. Learnlv1()

功能：LVQ1 权值学习函数

格式：(1)　[dW，LS] = learnlv1(W，P，Z，N，A，T，E，gW，gA，D，LP，LS)

　　　(2)　info = learnlv1(code)

说明：式中，dW 为 S*R 权值（或阈值）变化矩阵；LS 为当前学习状态(可省略)；W 为 S*R 的权值矩阵或者为 S*1 的阈值矢量；P 为 R*Q 的输入矢量或者为 1*Q 的全为 1 的矢量；Z 为 S*Q 的输入层的权值矢量(可省略)；N 为 S*Q 的网络输入矢量(可省略)；A 为 S*Q 的输出矢量；T 为 S*Q 的目标输出矢量(可省略)；E 为 S*Q 误差矢量 (可省略)；gW 为 S*R 的与性能相关的权重梯度矩阵(可省略)；gA 为 S*Q 的与性能相关的输出梯度值矩阵(可省略)；D 为 S*S 的神经元距离矩阵(可省略)；LP 为学习参数，该函数的学习参数由 LP.lr 构成，缺省值为 0.01；LS 为学习函数声明(可省略)。函数 learnk(code)返回学习函数的有关信息，code 字符串的取值为：

pnames——返回学习函数参数的名称；

pdefaults——返回学习函数参数的缺省值；

needg——当该函数使用了参数 gW 和 gA 时返回值为 1。

Learnlv1 函数利用 LVQ1 规则对权值进行调整，该函数多用于学习矢量量化网络中。在 learnlv1 函数中，当网络输出为 0 时，权值不作调整；当输出不为 0 时，权值将根据梯度矢量 gA 的取值进行调整：

4. Plotvec()

功能：用不同颜色绘制矢量的函数

格式：plotvec(X，C，M)

说明：式中 X 为一个列矢量矩阵；C 为标记颜色坐标的行矢量；M 为指定绘图时矢量的标记符号，缺省值为‘+’。

例子 2-6 针对一组输入向量，设计一个 LVQ 神经网络，经过训练后，能对给定数据进行模式识别。

MATLAB 代码如下：

```
%输入向量 P 及其对应的类别向量 C
P=[-6 -4 -2   0 0 0 0 2 4 6; 0 2 -2 1 2 -2 1 2 -2 0];
C=[1 1 1 2 2 2 2 1 1 1];
%将类别向量 C 转换为目标向量 T
```

```
T=ind2vec(C);
%绘制输入向量 P，如图 2-7-2 所示，用颜色将输入向量分为两类
plotvec(P,C,'*r');
%输入向量绘制在一个横坐标在[-8 8]之间，纵坐标在 [-3 3]之间的坐标平面内
axis([-8 8 -3 3]);
%创建一个 LVQ 神经网络，隐含层有 5 个神经元，[0.6 0.4]表示在隐含层的权值中，有 60%的
%列的第一行的值为 1，40%的列的第一行值为 1，也就是说有 60%的列属于第一类，40%属于
%第二类，网络的其他参数取默认值
net = newlvq(minmax(P),5,[0.6 0.4]);
net.trainParam.epochs=100;
net=train(net,P,T);
%给定数据，输出网络的分类结果测试网络的性能
p=[0 1;0.2 0];
y=sim(net,p);
yc=vec2ind(y)
```

程序运行后，得到图 2-7-2 和图 2-7-3，由图 2-7-3 可以看出网络在训练到第 4 次时已经达到误差要求，在命令行窗口中得到下面结果：

```
%用 TRAINR 作为训练函数，最大训练次数为 100 次
 >> TRAINR, Epoch 0/100
 TRAINR, Epoch 4/100
 TRAINR, Performance goal met.
%对给定数据，一个归于第 2 类，一个归于第 1 类
 yc =
      2      1
```

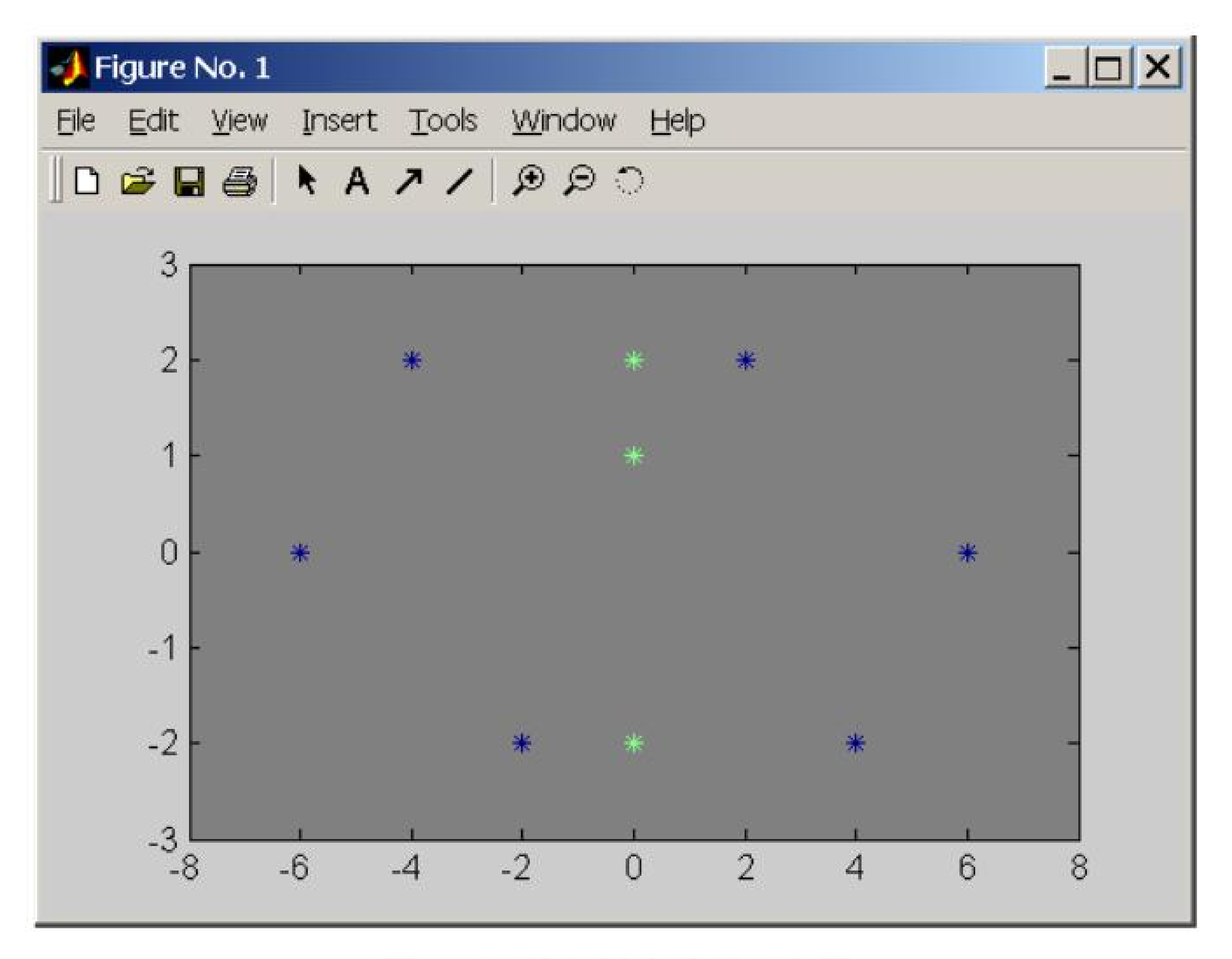


图 2-7-2 输入样本数据分布图

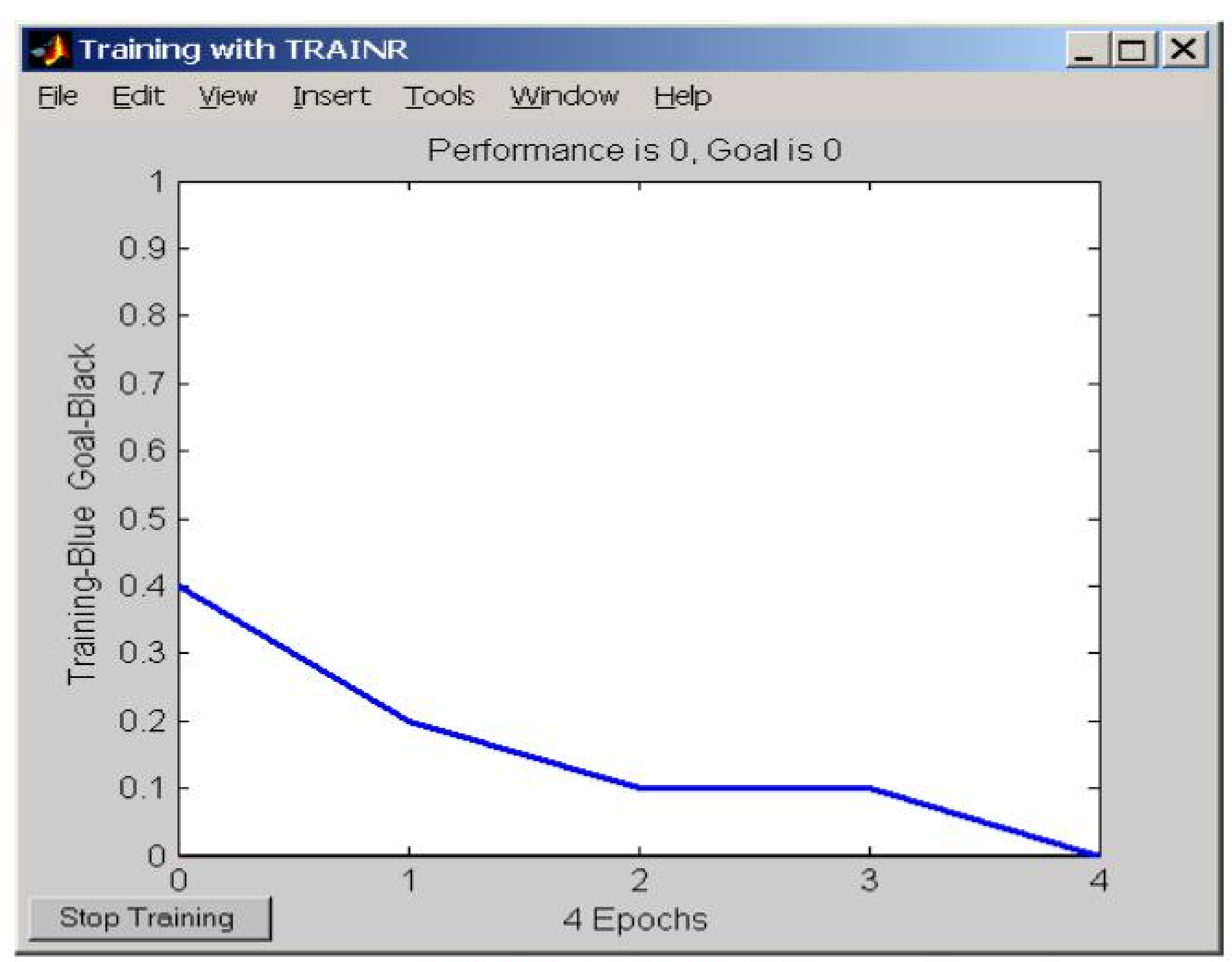


图 2-7-3 训练误差曲线

由类别向量 C 可以看出，输入样本数据中，前 3 个和后 3 个属于一类，中间 4 个属于第二类，由绘制的输入向量图，也看到用颜色将其分为了二类，输入测试数据 p=[0 1;0.2 0]后，网络输出结果表明其分类是正确的。

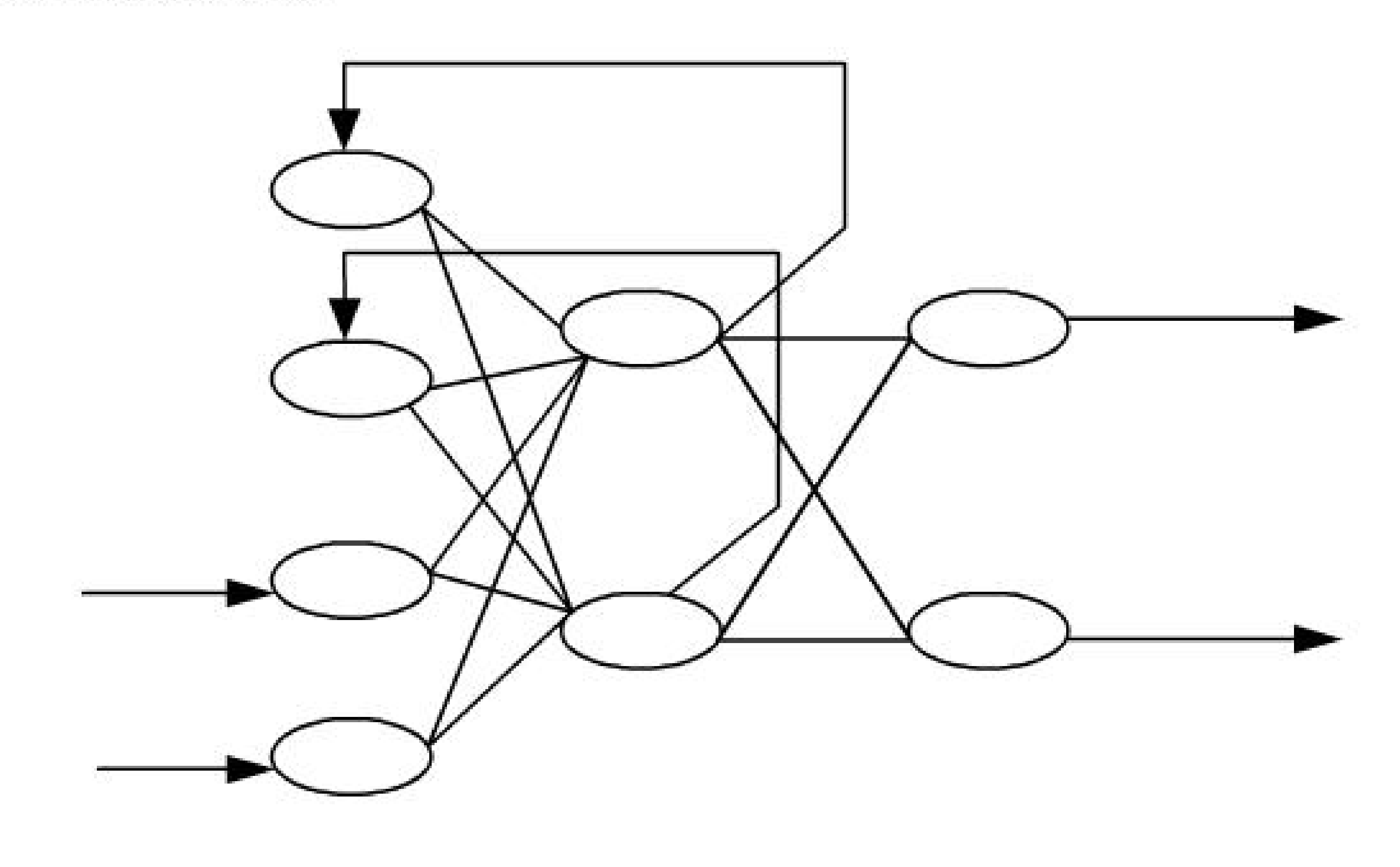
2.8 Elman 神经网络算法模型与学习算法

2.8.1Elman 神经网络结构

与前述神经网络不同，Elman 神经网络是一种反馈神经网络，比前向神经网络具有更强的计算能力，前向神经网络比较重视学习算法的研究，而反馈神经网络的稳定性相对于学习算法来说更重要。

图 2-8-1 给出了 Elman 神经网络的结构，该网络模型是 Elman 于 1990 年提出来的，具有与 MLP 网络相似的多层结构。Elman 网络由四层组成，除了普通的隐含层外，还有一个特别的隐含层，称为承接层，也称为上下文层或状态层。输入层的神经元仅起信号传输作用，输出层神经元起线性加

权作用。承接层从隐含层接收反馈信号，用来记忆隐含层神经元前一时刻的的输出值，承接层神经元的输出经延迟与存储，再输入到隐含层。这样就使其对历史数据具有敏感性，增加了网络自身处理动态信息的能力，从而达到动态建模的目的。如果只有正向连接是适用的，而反馈连接被预定为恒值，则网络可视为普通的前馈网络。



-16 Elman

2.8.2Elman 神经网络学习算法.

Elman 神经网络算法可以采用 BP 网络中所使用的附加动量的梯度下降反向传播算法对网络进行训练，下面给出 Elman 神经网络各层之间的表达式和误差函数，详细算法请参照 3.1.2 中的 BP 网络学习算法的附加动量改进算法。

在图 2-8-1 中，如果输入向量 $\mathbf{u}$ 为 r 维向量，输出向量 $\mathbf{y}$ 为 m 维，隐含层输出向量 $\mathbf{x}$ 为 n 维，承接层输出向量 $\mathbf{x}_c$ 为 n 维，w^1, w^2, w^3 分别为隐含层到输出层、输入层到隐含层、承接层到隐含层的连接权值。$g(\cdot)$ 为输出神经元的激活函数，是隐含层输出的线性组合。$f(\cdot)$ 为隐含层神经元的激活函数，则各层之间的表达式如下：

$$y(k) = g(w^3 x(k))$$

$$x(k)=f(w^1 x_c(k)+w^2(u(k-1)))$$

$$x_c(k) = x(k-1)$$

网络权值修正方法采用 BP 算法中的方法，衡量算法是否结束的误差函数如下：

$$E = \sum_{k=1}^{n}[y(k)-d(k)]^2$$

式中 $d(k)$ 为期望输出向量。

2.8.3Elman 神经网络学习算法的 MATLAB 实现

表 2-8-1 列出了 MATLAB 神经网络工具箱中提供的与算法相关的 Elman 神经网络工具函数和基本功能。在 MATLAB 的命令行窗口中输入“help elman”，便可得到与 Elman 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍

表 2-8-1 Elman 神经网络的重要函数和基本功能

函数名	功能
newelm()	生成一个 Elman 神经网络
trains()	根据已设定的权值和阈值对网络进行顺序训练
traingdx()	自适应学习速率动量梯度下降反向传播训练函数
learngdm()	动量梯度下降权值和阈值学习函数

下面对表 2-8-1 中的工具函数的使用进行说明，并通过一个预测空调负荷的实例来说明如何利用工具函数建立一个 Elman 神经网络。

例子 2-7 表 2-8-2 为某单位办公室七天上午 9 点到 12 点的空调负荷数据，数据已经做了归一化处理，预测方法采用前 6 天的数据作为网络的训练样本，每 3 天的负荷作为输入向量，第 4 天的负

表 2-8-2 空调负荷数据表

时间	9 时负荷	10 时负荷	11 时负荷	12 时负荷
第 1 天	0.4413	0.4707	0.6953	0.8133
第 2 天	0.4379	0.4677	0.6981	0.8002

第 3 天	0.4517	0.4725	0.7006	0.8201
第 4 天	0.4557	0.4790	0.7019	0.8211
第 5 天	0.4601	0.4811	0.7101	0.8298
第 6 天	0.4612	0.4845	0.7188	0.8312
第 7 天	0.4615	0.4891	0.7201	0.8330

荷作为目标向量，第七天的数据作为网络的测试数据。

完整的 MATLAB 代码如下：

```
%根据预测方法得到输入向量和目标向量
P=[0.4413 0.4707 0.6953 0.8133 0.4379 0.4677 0.6981 0.8002 0.4517 0.4725 0.7006 0.8201;
0.4379 0.4677 0.6981 0.8002 0.4517 0.4725 0.7006 0.8201 0.4557 0.4790 0.7019 0.8211;
0.4517 0.4725 0.7006 0.8201 0.4557 0.4790 0.7019 0.8211 0.4601 0.4811 0.7101 0.8298;]';
T=[0.4557 0.4790 0.7019 0.8211;
0.4601 0.4811 0.7101 0.8298;
0.4612 0.4845 0.7188 0.8312]';
%输入向量的取值范围为[0 1]，用 threshold 来标记
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
%创建一个 Elman 神经网络，隐含层的神经元个数为 17 个，4 个输出层神经元，隐含层激活函
%数为 tansig，输出层激活函数为 purelin
net=newelm(threshold,[17,4],{'tansig','purelin'});
net.trainParam.epochs=3000;
net=init(net);
net=train(net,P,T);
%输入测试数据
P_test=[0.4557 0.4790 0.7019 0.8211 0.4601 0.4811 0.7101 0.8298 0.4612 0.4845 0.7188 0.8312]';
T_test=[0.4615 0.4891 0.7201 0.8330]';
y=sim(net,P_test)
%在测试数据下，计算网络的输出和实际目标向量之间的差值
error=y-T_test;
%在坐标平面上画出差值曲线
plot(1:4,error,'-');
```

程序运行后，得到图 2-8-2，命令行窗口中的结果如下：

```
……
TRAINGDX, Epoch 2950/3000, MSE 4.53259e-005/0, Gradient 0.000187623/1e-006
```

TRAINGDX, Epoch 2975/3000, MSE 4.4287e-005/0, Gradient 0.00424495/1e-006
TRAINGDX, Epoch 3000/3000, MSE 4.35966e-005/0, Gradient 0.00219681/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.

y =

    0.4776
    0.5052
    0.7173
    0.8189

由输出结果可以看出，经过 3000 次训练后，网络训练误差为 2.6293e-005，虽然没有达为到目标误差的默认值 0，不过已经足够小了。由输出的 y 值与实际负荷值的比较和误差曲线图，网络的预报误差是比较小的，只是在个别时间出现相对较大的误差，这是因为训练样本数据太小所造成的。

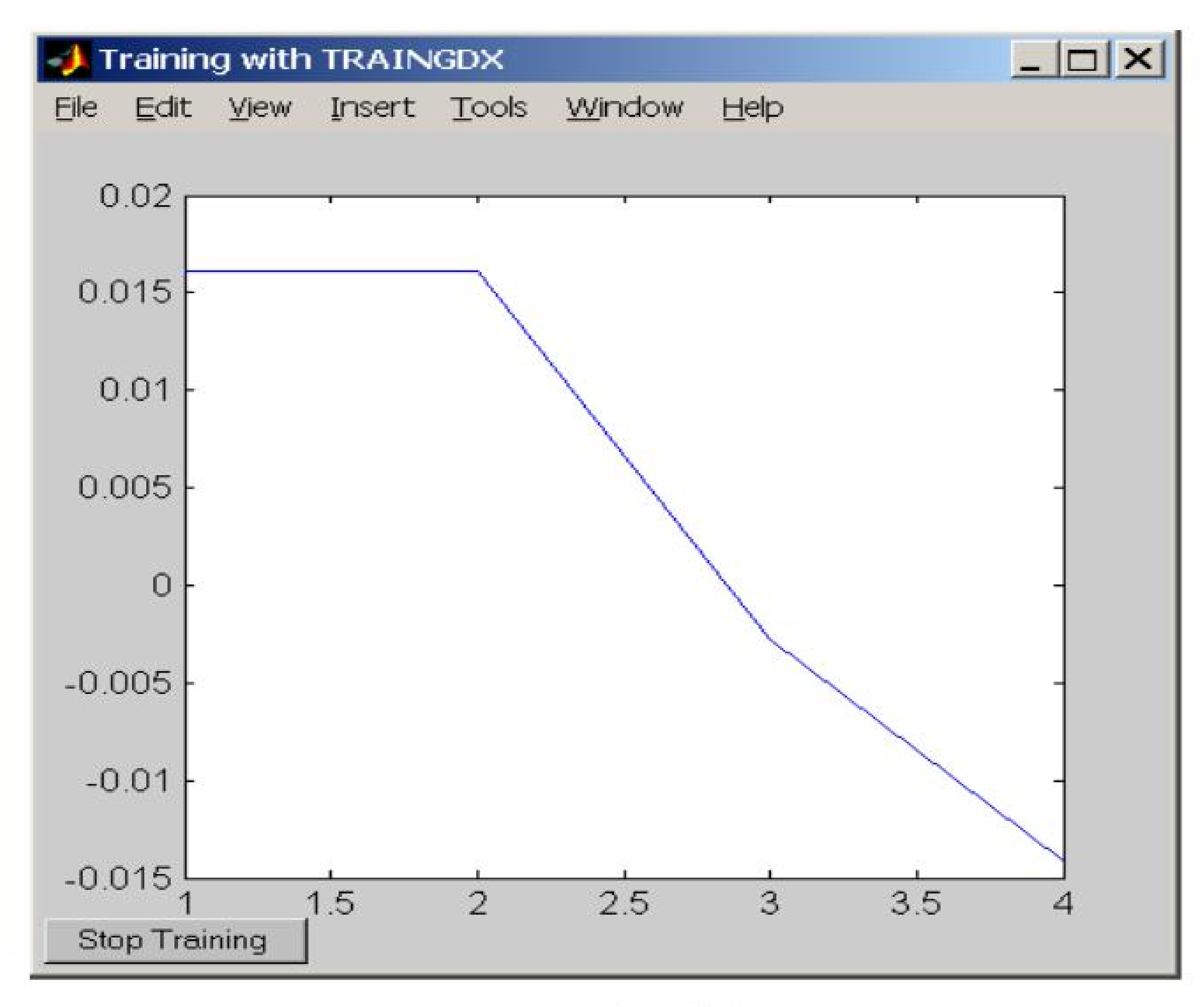


图 2-8-2 训练误差曲线

2.9 Hopfield 神经网络模型与学习算法

Hopfield 网络是神经网络发展历史上的一个重要的里程碑。Hopfield 神经网络是 1982 年美国物理学家 J.Hopfield 首先提出来的，属于反馈神经网络类型。与前向型神经网络不同，前向神经网络不考虑输出与输入之间在时间上的滞后影响，其输出与输入之间仅仅是一种映射关系。而 Hopfield 网络则不同，它采用反馈连接，考虑输出与输入在时间上的传输延迟，所表示的是一个动态过程，需要用差分或微分方程来描述，因而 Hopfield 网络是一种由非线性元件构成的反馈系统，其稳定状态的分析比前向神经网络要复杂得多。

Hopfield 用能量函数的思想形成了一种新的计算方法，阐明了神经网络与动力学的关系，并用非线性动力学的方法来研究这种神经网络的特性，建立了神经网络稳定性判据，并指出信息存储在网络各个神经元之间的连接上，形成了所谓的 Hopfield 网络。 Hopfield 还将该反馈网络同统计物理中的 lsing 模型相类比，把磁旋的向上和向下方向看成神经元的激活和抑制两种状态，把磁旋的相互作用看成神经元的突触权值。这种类推为大量的物理学理论和许多的物理学家进入神经网络领域铺平了道路。1984 年，Hopfield 设计并研制了 Hopfleld 网络模型的电路，指出神经元可以用运算放大器来实现，所有神经元的连接可用电子线路来模拟，称之为连续 Hopfield 网络。使用该电路，Hopfleld 成功地解决了旅行商(TSP)计算难题(优化问题)。

2. 9. 1 离散 Hopfield 神经网络

1982 年 Hopfield 提出离散的 Hopfield 网络同前向神经网络相比，在网络结构、学习算法和运行规则上都有很大的不同。

1.离散 Hopfield 网络模型

离散 Hopfield 网络是单层全互连的，其表现形式如图 2-9-1、2-9-2 所示的两种形式。

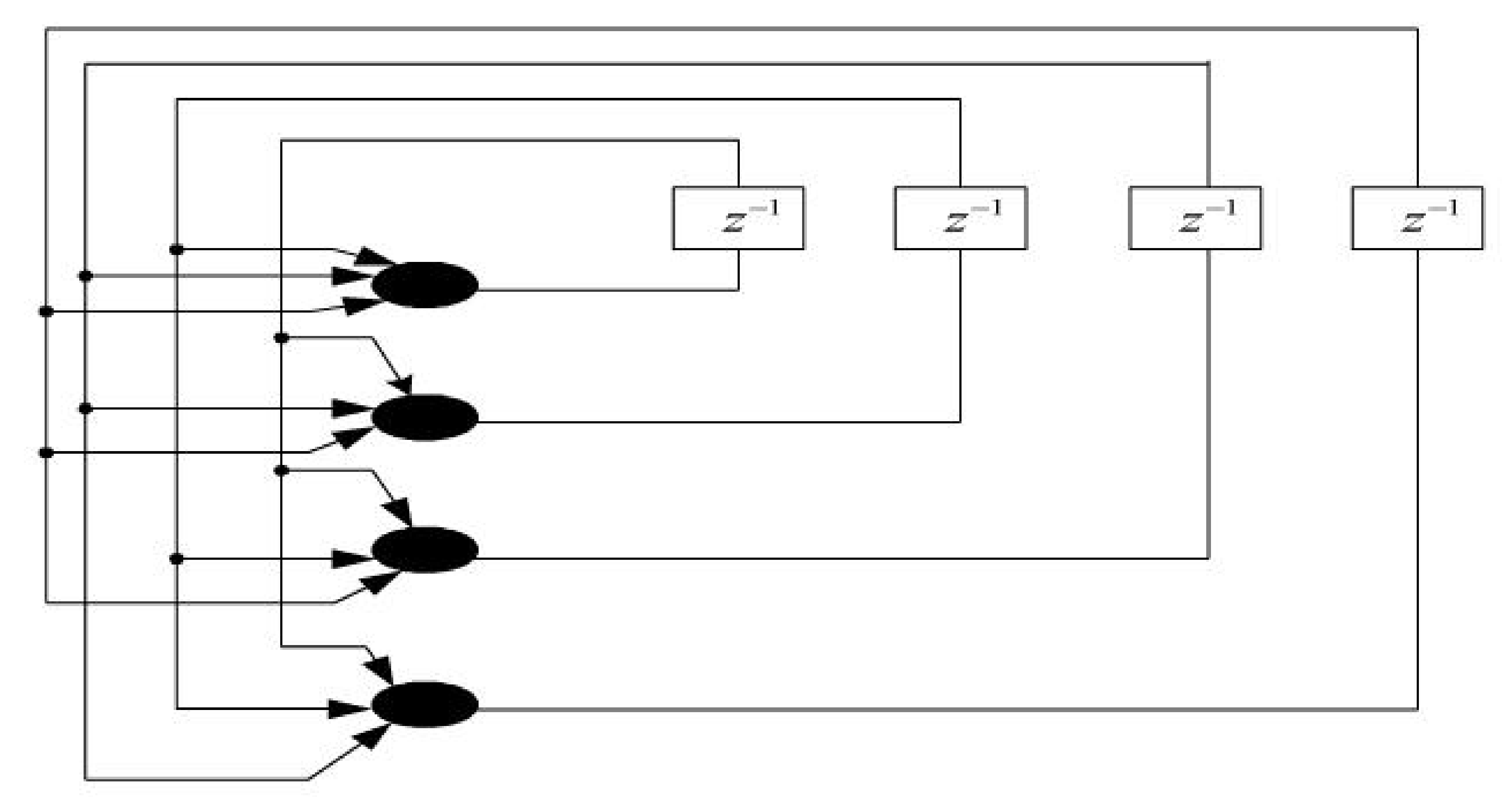
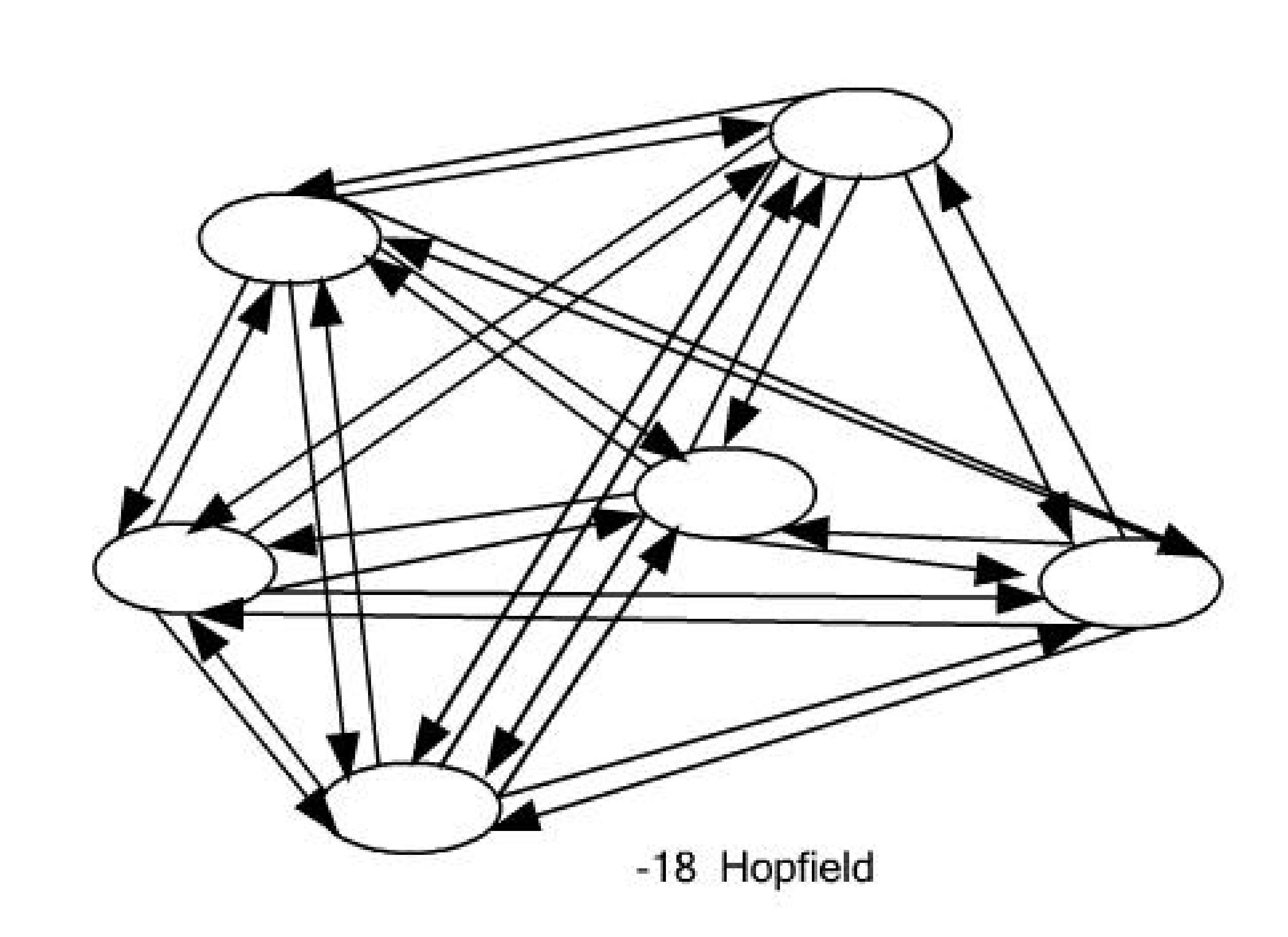


图 2-17 Hopfield神经网络结构



-18 Hopfield

神经元可取一位（0/1）或(-1/1)，其中的任意神经元 i 与 j 间的突触权值为 w_{ij}，神经元之间联接是对称的，即 $w_{ij}=w_{ji}$，神经元自身无联接，即 w_{ii}=0。虽然神经元自身无联接，但每个神经元都同其它的神经元相连，即每个神经元都将其输出通过突触权值传递给其它的神经元，同时每个神经元又都接收其它神经元传来的信息，这样对于每个神经元来说，其输出信号经过其它神经元又有可能反馈给自己，所以 Hopfield 网络是一种反馈神经网络。

设 Hopfield 网络中有 n 个神经元，其中任意神经元 i 的输入用 u_i 表示，输出用 v_i 表示，它们都是时间的函数，其中 $v_i(t)$ 也称为神经元 i 在 t 时刻的状态。

$$v_i(t)=\sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}u_j(t)+b_i \tag{2-7}$$

上式中的b_i表示神经元i的阈值或偏差。相应神经元i的输出或状态为

$$v_i(t+1)=\mathrm{f}(\mathrm{u}_i(t))$$

其中的激励函数$\mathrm{f}(\cdot)$可取阶跃函数$\mathrm{u}(t)$或符号函数$\mathrm{sgn}(t)$)。如取符号函数，则 Hopfield 网络的神经元的输出$v_i(t+1)$取离散值 1 或-1，即

$$v_i(t+1)=\begin{cases}1 & \sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}v_j(t)+b_i\geq 0\\ -1 & \sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}v_j(t)+b_i<0,\end{cases} \tag{2-8}$$

2.离散 Hopfield 网络的运行规则

Hopfield 网络按动力学方式运行，其工作过程为状态的演化过程，即从初始状态按“能量”(Lyapunov 函数)减小的方向进行演化，直到达到稳定状态，稳定状态即为网络的输出。

Hopfield 网络的工作方式主要有两种形式：

(1)串行(异步)工作方式 在任一时刻t，只有某一神经元i (随机的或确定的选择)依上式变化，而其它神经元的状态不变。

(2)并行(同步)工作方式 在任一时刻t，部分神经元或全部神经元的状态同时改变。

下面以串行(异步)工作方式为例说明 Hopfield 网络的运行步骤：

第一步 对网络进行初始化；

第二步 从网络中随机选取一个神经元i；

第三步 按公式(2-7)求出该神经元 i 的输入$u_i(t)$；

第四步 按公式(2-8)求出该神经元 i 的输出 $v_i(t+1)$，此时网络中的其他神经元的输出保持不变；

第五步 判断网络是否达到稳定状态，若达到稳定状态或满足给定条件则结束；否则转到第二步继续运行。

这里网络的稳定状态定义为：若网络从某一时刻以后，状态不再发生变化，则称网络处于稳定状态。

$$v(t+\Delta t)=v(t) \qquad \Delta t>0$$

Hopfield 网络存在稳定状态，则要求 Hopfield 网络模型满足如下条件：网络为对称连接，即 $w_{ij}=w_{ji}$；神经元自身无连接即 w_{ii}。这样 Hopfield 网络的突触权值矩阵 w 为零对角对称矩阵。

在满足以上参数条件下，Hopfield 网络“能量函数”(Lyapunov 函数)的“能量”在网络运行过程中应不断地降低，最后达到稳定的平衡状态。

Hopfield 网络的“能量函数”定义为

$$\mathrm{E}=-\frac{1}{2}\sum_{\substack{i=1\\i\neq j}}^{\mathrm{n}}\sum_{\substack{j=1\\j\neq i}}^{\mathrm{n}} w_{ij}v_i v_j+\sum_{i=1}^{\mathrm{n}} b_i v_i$$

Hopfield 反馈网络是一个非线性动力学系统，Hopfield 网络按动力学方式运行，即按“能量函数”减小的方向进行演化，直到达到稳定状态。因而上式所定义的“能量函数”值应单调减小。为说明这一问题，可考虑网络中的任意神经元，其能量函数为

$$\mathrm{E}_i=-\frac{1}{2}\sum_{\substack{i=1\\i\neq j}}^{\mathrm{n}} w_{ij}v_i v_j+b_i v_i$$

从 t 时刻至 $t+1$ 时刻的能量变化量为

$$\begin{aligned}\Delta E_i &= E_i(t+1) - E_i(t) \\ &= -\frac{1}{2}\sum_{\substack{i=1 \\ i\neq j}}^{n} w_{ij} v_i(t+1) v_j + b_i v_i(t+1) + \frac{1}{2}\sum_{\substack{i=1 \\ i\neq j}}^{n} w_{ij} v_i(t) v_j - b_i v_i(t) \\ &= -\frac{1}{2}\left[v_i(t+1) - v_i(t)\right]\left[\sum_{\substack{i=1 \\ i\neq j}}^{n} w_{ij} v_j + b_i\right]\end{aligned}$$

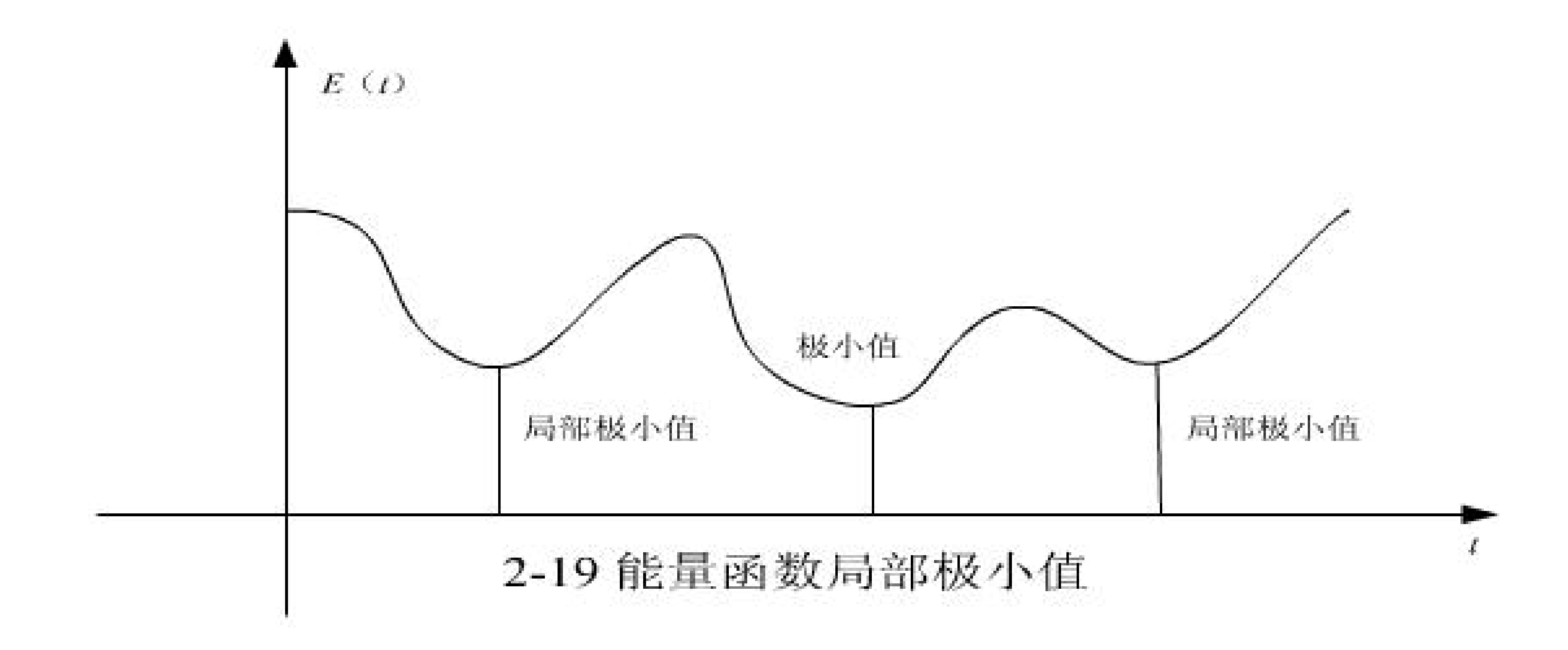
由上式可得：$\Delta E_i \leq 0$

因为神经元 i 为网络中的任意神经元，而网络中的所有神经元都按同一规则进行状态更新，所以网络的能量变化量应小于等于零，即

$$\Delta E \leq 0$$

$$E(t+1) \leq E(t)$$

所以在满足参数条件下，Hopfield 网络状态是向着能量函数减小的方向演化。由于能量函数有界，所以系统必然会趋于稳定状态，该稳定状态即为 Hopfield 网络的输出。能量函数的变化曲线如图 2-9-3 所示，曲线含有全局最小点和局部最小点。将这些极值点作为记忆状态，可将 Hopfield 网络用于联想记忆；将能量函数作为代价函数，全局最小点看成最优解，则 Hopfield 网络可用于最优化计算。



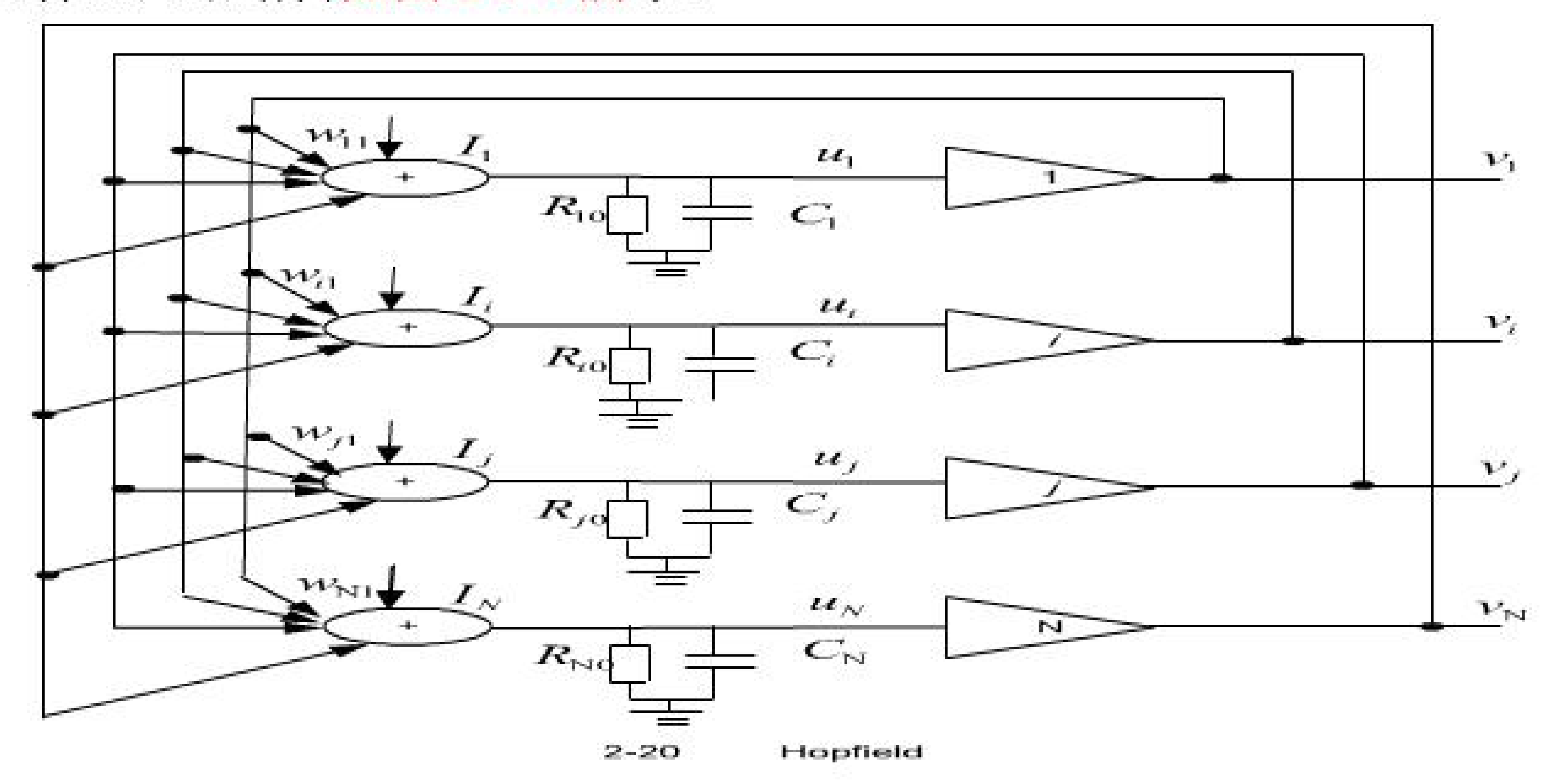
2-19 能量函数局部极小值

2.9.2 连续 Hopfield 神经网络

1984 年 Hopfield 采用模拟电子线路实现了连续 Hopfield 网络，因为该网络中神经元的激励函数为连续函数，所以该网络也被称为连续 Hopfield 网络。在连续 Hopfield 网络中，网络的输入、输出均为模拟量，各神经元采用并行(同步)工作方式。利用这一特征 Hopfield 将该网络应用于优化问题的求解上，并成功地解决了 TSP 问题。

1.连续 Hopfield 网络模型

连续 Hopfield 神经网络结构如图 2-9-4 所示：



2-20 Hopfield

从图 2-9-4 可见 Hopfield 神经网络中的每个神经元都是由运算放大器及其相关的电路组成，其中任意一个运算放大器 i(或神经元 j)都有两组输入：第一组是恒定的外部输入，用 I_j 表示，这相当于放大器的电流输入；第二组是来自其他运算放大器的反馈连接，这相当于神经元 i 与神经元 j 之间的连接权值。u_i 表示运算放大器 i 的输入电压，v_i 表示运算放大器 i 的输出电压，它们之间的关系为

$$v_i = \mathrm{f}(u_i)$$

其激励函数 $\mathrm{f}(\cdot)$ 常取 Sigmoid 型函数中双曲线正切函数，即

$$v_i = \mathrm{f}(u_i) = \tanh(\frac{a_i u}{2}) = \frac{1-\exp(-a_i u_i)}{1+\exp(-a_i u_i)}$$

其中 $\frac{a_i}{2}$ 为曲线在原点处的斜率，即

$$\frac{a_i}{2} = \frac{d\,\mathrm{f}(u_i)}{du_i} u_i = 0$$

因此，称 a_i 为运算放大器 i（或神经元 i）的增益。

激励函数 f() 的反函数为 $\mathrm{f}^{-1}()$ 为：

$$u_i = \mathrm{f}^{-1}(v_i) = -\frac{1}{a_i}\log\left(\frac{1-v_i}{1+v_i}\right)$$

对于图 2-9-4 所示的连续 Hopfield 神经网络模型，根据基尔霍夫电流定律有

$$C_i \frac{du_i}{dt} + \frac{u_i}{R_{io}} = \sum_{j=1}^{\mathrm{N}} \frac{1}{R_{ij}}(v_j - u_i) + I_i$$

$$C_i \frac{du_i}{dt} = \sum_{j=1}^{\mathrm{N}} w_{ij}(v_j - u_i) + I_i - \frac{u_i}{R_{io}}$$

其中的 $w_{ij} = \frac{1}{R_{ij}}$，设 $\frac{1}{R_{ij}} = \frac{1}{R_{io}} + \sum_{j=i}^{\mathrm{N}} w_{ij}$，则上式可改写为：

$$C_i \frac{du_i}{dt} = \sum_{j=1}^{\mathrm{N}} w_{ij} v_j + I_i - \frac{u_i}{R_i}$$

与离散 Hopfield 神经网络相同，连续 Hopfield 网络的突触权值是对称的，且无自反馈，即：$w_{ij} = w_{ji}$，$w_{ii} = 0$。对于连续 Hopfield 神经网络模型，其能量函数定义如下

$$\mathrm{E} = -\frac{1}{2}\sum_{i=1}^{\mathrm{n}}\sum_{j=1}^{\mathrm{n}} w_{ij} v_i v_j + \sum_{i=1}^{\mathrm{N}} \frac{1}{R_i}\int_0^{u_i} \mathrm{f}^{-1}(v_i)\,dv_i - \sum_{i=1}^{\mathrm{N}} I_i v_i$$

2.连续 Hopfield 网络稳定性分析

为证明该能量函数 E 是单调下降的，可求上式对时间 t 的微分，即

$$\frac{d\,\mathrm{E}}{dt}=\frac{d\,\mathrm{E}}{dv_i}-\frac{dv_i}{dt}=-\sum_{i=1}^{\mathrm{N}}\left(\sum_{j=1}^{\mathrm{N}}w_{ij}v_j-\frac{u_i}{R_j}+I_i\,\frac{dv_i}{dt}\right)$$

将式 $C_i\,\frac{du_i}{dt}=\sum_{j=1}^{\mathrm{N}}w_{ij}v_j+I_i-\frac{u_i}{R_i}$ 代入上式可得

$$-\frac{d\,\mathrm{E}}{dt}=-\sum_{i=1}^{\mathrm{N}}C_i(\frac{du_i}{dt})\frac{dv_i}{dt}$$

$$=-\sum_{i=1}^{\mathrm{N}}C_i\,\frac{d\,\mathrm{f}^{-1}(v_i)}{dt}\frac{dv_i}{dt}$$

$$=-\sum_{i=1}^{\mathrm{N}}C_i\,\frac{d\,\mathrm{f}^{-1}(v_i)}{dv_i}\frac{dv_i}{dt}$$

$$=-\sum_{i=1}^{\mathrm{N}}C_i\,\frac{d\,\mathrm{f}^{-1}(v_i)}{dv_i}(\frac{dv_i}{dt})^2$$

因为 $\mathrm{f}^{-1}(v_i)$ 为单调递增函数，所以

$$\frac{d\,\mathrm{f}^{-1}(v_i)}{dv_i}\geq 0,\text{又}\left[\frac{dv_i}{dv_i}\right]^2\geq 0, C_i\geq 0,$$

因而有：

$$\frac{d\,\mathrm{E}}{dt}\leq 0$$

由于能量函数 E 是有界的，因此，连续 Hopfield 网络模型是稳定的。

连续 Hopfield 网络模型对生物神经元模型做了大量的简化，但仍突出了生物系统神经计算的主要特性，如：

1)连续 Hopfield 网络的神经元作为 I/O 变换，其传输特性具有 Sigmoid 特性；

2)具有时空整合作用；

3)在神经元之间存在着大量的兴奋性和抑制性联接，这种联接主要是通过反馈来实现；

4)具有既代表产生动作电位的神经元，又有代表按渐进方式工作的神经元，即保留了动态和非线性两个最重要的计算特性。

Hopfield 神经网络经常用于存储一个或者多个稳定的目标向量。当向网络提供输入向量时，存储在网络中的接近于输入的目标向量就被唤醒。Hopfield 神经网络设计的目标就是使得网络存储一些特定的平衡点，当给定网络一个初始条件时，网络最后会在这样的点上停下来。由于输出又被反馈到输入，所以一旦网络开始运行，整个网络就是递归的。

2.9.3 Hopfield 神经网络的 MATLAB 实现

表 2-9-1 列出了 MATLAB 神经网络工具箱提供的与算法相关的 Hopfield 神经网络工具函数和基本功能。在 MATLAB 工作空间的命令行输入“help hopfield”，便可得到与 Hopfield 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍

表 2-9-1　Hopfield 网络的重要函数和基本功能

函数名	功能
satlin( )	饱和线性传递函数
satlins( )	对称饱和线性传递函数
newhop( )	生成一个 Hopfield 回归网络
nnt2hop( )	更新 NNT 2.0 Hopfield 回归网络

下面对表 2-9-1 中的工具函数的使用进行说明，并通过一个识别印刷体数字的实例来说明如何利用工具函数建立一个 Hopfield 神经网络。

1. Newhop( )

功能：生成一个 Hopfield 回归网络。

格式：　net = newhop(T)

说明：式中 net 为生成的神经网络，具有在 T 中的向量上稳定的点；T 是具有 Q 个目标向量的 R*Q 矩阵（元素必须为-1 或 1）。Hopfield 神经网络经常被应用于模式的联想记忆中。Hopfield 神经网络仅有一层，其激活函数用 satlins( )函数，层中的神经元有来自它自身的连接权和阈值。

2. Satlins( )

功能：对称饱和线性传递函数

格式：　A = satlins(N)

A 输出向量矩阵；N 是由网络的输入向量组成的 S*Q 矩阵，返回的矩阵 A 与 N 的维数大小一致，A 的元素取值位于区间[0，1]内。当 N 中的元素介于-1 和 1 之间时，其输出等于输入；当输入值小于-1 时返回-1；当输入值大于 1 时返回 1。

例子 2-8 设印刷体数字由 10×10 点阵构成，就是将数字分成很多小方块，每个方块就对应数字的一部分，构成数字本部分的方块用 1 表示，空白处用-1 表示。试设计一个 Hopfield 网络，能够正确识别印刷体的数字。

从简洁的角度出发，下面将只给出 1 和 2 的点阵表示形式，完整的 MATLAB 代码如下：

```
%数字 1 的点阵表示
one=[-1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1    -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1
-1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1
1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1];
%数字 2 的点阵表示
two=[1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 1
1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 1 1 1 -1
-1 1 1 1 1 1 1 1 1 -1 -1];
%设定网络的目标向量
T=[one;two]';
%创建一个 Hopfield 神经网络
net=newhop(T);
%给定一个受噪声污染的数字 2 的点阵，所谓噪声是指数字点阵中某些本应为 1 的方块变成了
%-1
no2={[1 1 1 -1 1 1 -1 1 -1 -1    1 1 1 1 1 1 1 1 -1 -1    -1 1 -1 1 -1 -1 1 1 -1 -1 -1 -1 1 -1 1 -1 1 1 -1 -1 1
1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1    1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 1 1 1 1
-1 -1 1 1 -1 1 1 1 1 1 1 -1 -1]'};
```

%对网络进行仿真，网络的仿真步数为 5

tu2 =sim(net,{1,5},{},no2)

%输出仿真结果向量矩阵中的第 3 列向量，并将其转置

tu2{3}'

程序运行后，在命令行窗口得到下面的结果：

```
ans =

  Columns 1 through 21

     1     1     1     1     1     1     1     1    -1    -1     1     1     1
     1     1     1     1     1    -1    -1    -1

  Columns 22 through 42

    -1    -1    -1    -1    -1     1     1    -1    -1    -1    -1    -1    -1    -1
    -1     1     1    -1    -1     1     1

  Columns 43 through 63

     1     1     1     1     1     1    -1    -1     1     1     1     1     1
     1     1     1    -1    -1     1     1    -1

  Columns 64 through 84

    -1    -1    -1    -1    -1    -1    -1     1     1    -1    -1    -1    -1    -1
    -1    -1    -1     1     1     1     1

  Columns 85 through 100

     1     1     1     1    -1    -1     1     1     1     1     1     1     1     1
    -1    -1
```

由结果可以看出所得到的点阵与数字 2 的正常点阵是一致的，表明网络可以从受到污染的数字 2 的点阵中识别出数字 2，证明网络是有效的。

2.10 Boltzmann 神经网络模型与学习算法

20 世纪 80 年代，Ackley、Hinton 和 Sejnowski 等人以模拟退火思想为基础，对 Hopfield 模型引入了随机机制，提出了 Boltzmann 机。Boltzmann 机是第一个受统计力学启发的多层学习机，它是一类典型的随机神经网络，属于反馈神经网络类型。其命名来源于 Boltzmann 在统计热力学中的早期工作和网络本身的动态分布行为(其平衡状态服从 Boltzmann 分布)，并且 Boltzmann 机的运行机制服从模拟退火算法。Boltzmann 机结合 BP 网络和 Hopfield 网络在网络结构、学习算法和动态运行机制的优点，具有多层网络含义的网络结构、简易而有效的学习算法以及依概率方式工作的动态运行机制。

2.10.1Boltzmann 机的网络结构

Boltzmann 机网络结构如图 2-10-1 所示。Boltzmann 机由输入部、输出部和中间部构成。输入部和输出部神经元统称为显见神经元，是网络与外部环境进行信息交换的媒介，中间部的神经元称为隐见神经元，它们通过显见神经元与外界进行信息交换，但 Boltzmann 机网络没有明显的层次。另外，Boltzmann 机考虑到 Hopfield 网络的动态特性，其神经元是互联的，网络状态按照概率分布进行变化。

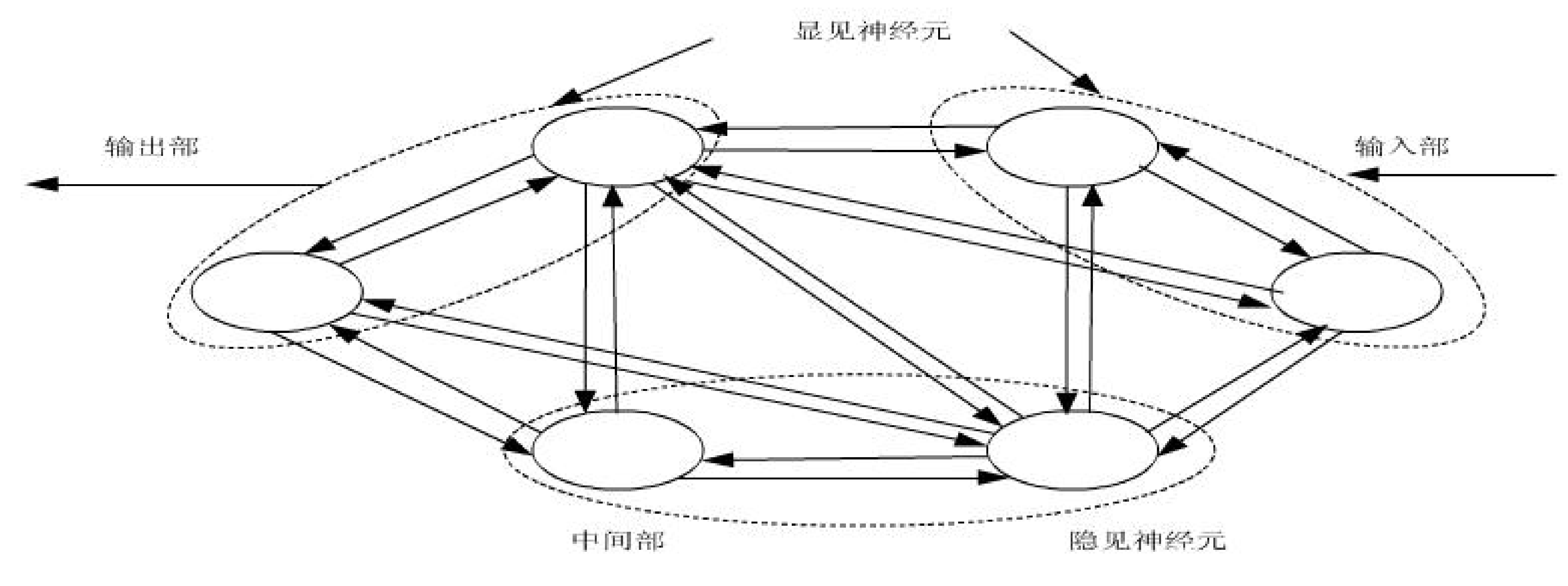


图2-21 Boltzmann机的网络结构

与 Hopfield 网络一样，网络中每一对神经元之间的信息传递是双向对称的，即 $w_{ij} = w_{ji}$，而且自身无反馈，即 $w_{ii} = 0$。学习期间，显见神经元将被外部环境“约束”在某一特定的状态，而中间部隐见神经元则不受外部环境约束。

Boltzmann 机中每个神经元的兴奋或抑制具有随机性，其概率取决于神经元的输入。Boltzmann 机中单个随机神经元的模型如图 2-10-2 所示。

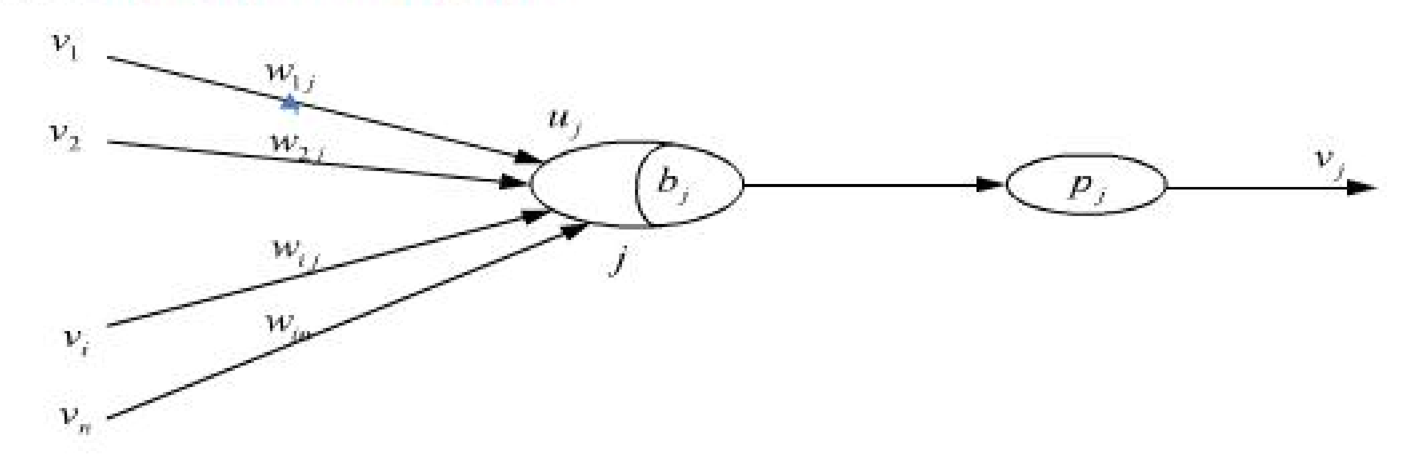


图 2-22 Boltzmann机神经元模型

图中 v_i 表示与神经元 j 相连接的神经元的输出，神经元 j 的全部输入信号的总和为 u_j，由下式给出。

$$u_j = \sum_{i}^{n} w_{ij} v_i + b_j$$

式中 b_j 是该神经元的阈值。将 b_j 看作连接权值为 1 的输入 v_0，可以将 b_j 归并到总的加权和中去，即得到下式：

$$u_j = \sum_{i=0}^{n} w_{ij} v_i \qquad (2\text{-}9)$$

神经元 j 的输出 v_j 依概率取 1 或 0：

v_j 取 1 的概率：$P(v_j = 1) = 1/(1 + e^{-u_j/T})$ （2-10）

v_j 取 0 的概率：$P(v_j = 0) = 1 - P(v_j = 1) = e^{-u_j/T}$ $P(v_j = 1$ （2-11）

显然，u_j 越大，则 v_j 取 1 的概率越大，而取 0 的概率越小，参数 T 称为“温度”，对 v_j 取 1 或 0 的概率有影响。在不同的温度下 v_j 取 1 的概率 P 随 u_j 的变化如图 2-10-3 所示。

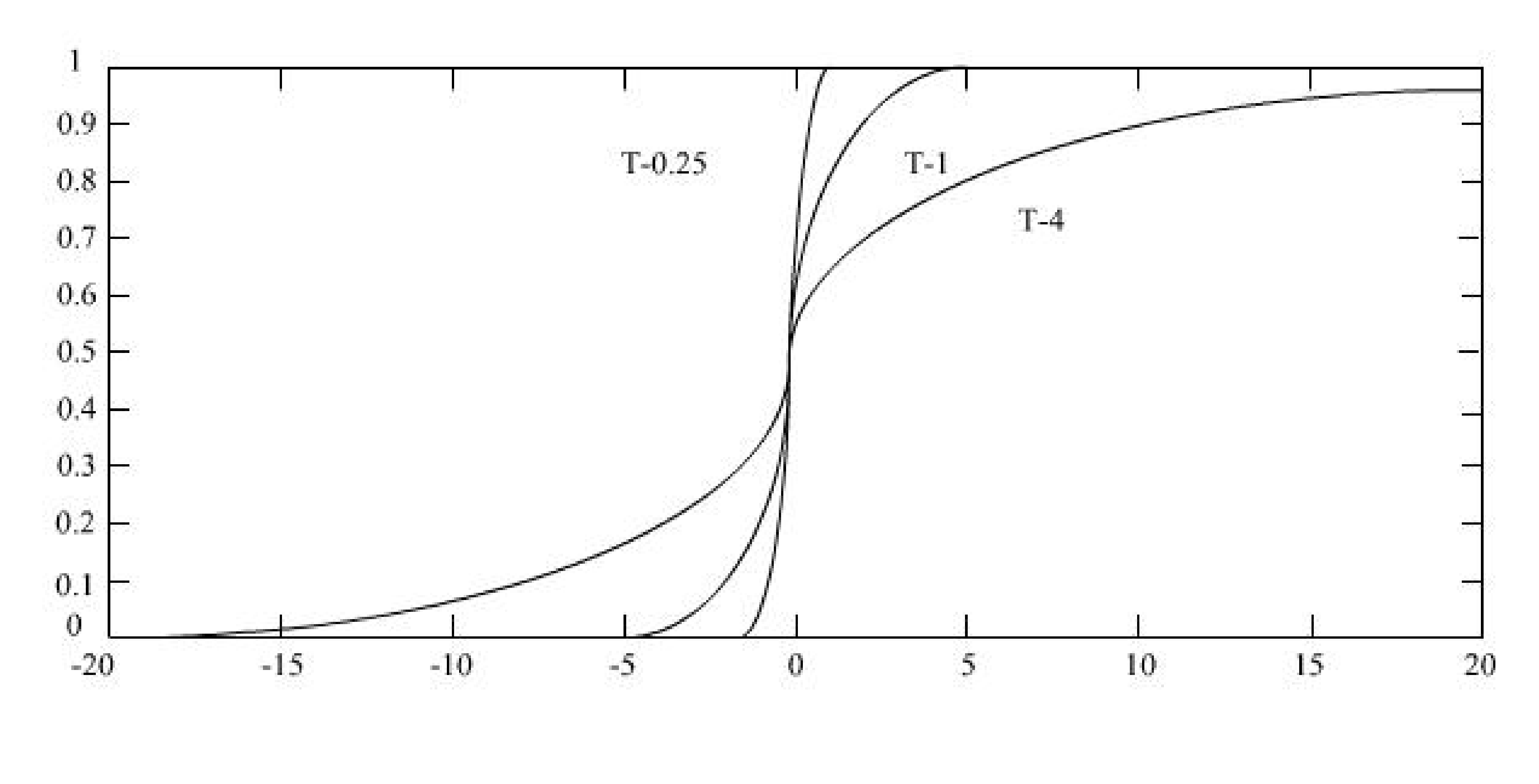


图 2-23　*P*~*U*关系曲线

从图 2-10-3 可见，T 越高时，曲线越平滑，因此，即使 u_j 有很大变动，也不会对 v_j 取 1 的概率变化造成很大的影响；反之，T 越低时，曲线越陡峭，当 u_j 有稍许变动时就会使概率有很大差异，当 T 趋向于 0 时，每个神经元不再具有随机特性，而且有确定的特性，激励函数变为阶跃函数，这时 Boltzmann 机趋向于 Hopfield 网络。从这个意义上来说，Hopfield 网络是 BoIbmann 机的特例。

2.10.2Boltzmann 机学习算法

将 Boltzmann 机视为一动力系统，利用公式可以证明能量函数的极小值对应系统的稳定平衡点，由于能量函数有界，当网络温度 T 以某种方式逐渐下降到某一特定值时，系统必趋于稳定状。将需要求解的优化问题的目标函数与网络的能量函数相对应，神经网络的稳定状态就对应优化目标的极小值。根据不同的用途，Boltzmann 的学习算法可以分为状态更新算法和联想记忆算法，前者主要用于解决优化组合问题，后者主要用于解决依依照一定概率重现记忆的问题。下面将只介绍应用比较多的状态更新算法。

设 Boltzmann 机网络有 N 个神经元，各神经元之间的连接权值为 $w_{ij}(i, j = 1, 2, \cdots$ ，各神经元的偏置值为 b_i ，输出为 v_i ，神经元 i 的内部状态为 H_i ，网络初始温度为 T_0 ，目标温度 T_d 。算法具体步骤如下：

1.网络初始化 给w_{ij}和b_i赋[-1，1]之间的随机数，置$w_{ij}=w_{ji}$，设定目标温度T_d的值。

2.求解内部状态 从N个神经元中随机选取一个神经元i，根据下式求解出神经元i的输入总和，即内部状态。

$$H_i(t)=\sum_{j=1,j\neq i}^{N} w_{ij}u_j(t)+b_j$$

3.更新神经元i状态 根据下面公式更新神经元i的状态：

$$P[v_i(t+1)=1]=\frac{1}{1+\exp(-H_i(t)/T)}$$

当$H_i(t)>0$时，直接令$v_i(t+1)=1$；当$H_i(t)<0$时，则产生一个位于[0,0.5]内的随机数$\varepsilon(t)$，如果$P[v_i(t+1)=1]>\varepsilon(t)$时，令$v_i(t+1)=1$，否则令$v_i(t+1)=v_i(t)$。

4.除i外的神经元的输出状态保持不变 即除i外的神经元的状态由下面公式求解得出：

$$v_j(t+1)=v_j(t) \qquad j=1,2,\cdots \qquad i$$

5.令$t=t+1$，按照下式计算出新的温度参数：

$$T(t+1)=\frac{T_0}{\log(t+1)}$$

6.第 5 步计算出的温度参数是否小于目标温度T_d，小于目标温度则算法结束，否则返回 2，进入下一轮计算。

在 MATLAB 的工具箱中，没有 Boltzmann 神经网络的相关函数，读者可以在学习第 4 章后，根据自己的需要，利用 MATLAB 中的数学计算函数来扩展 nnToolKit 工具包，实现 Boltzmann 神经网络的学习算法，完成自己特定的需要。

2.11 模糊神经网络

模糊神经网络是近年来智能控制与智能自动化领域的热点，美国早在 1988 年就召开了由NASA(国家航天航空局)主持的“神经网络与模糊系统”的国际研讨会，其后模糊神经网络的研究在美国、日本、法国、加拿大、新加坡等国蓬勃开展起来，成果大量涌现。

模糊系统是模糊数学在自动控制、信息处理、系统工程等领域的应用，属于系统论的范畴，而神经网络是人工智能的一个分支，属于计算机科学。模糊系统试图描述和处理人的语言和思维中存在的模糊性概念，从而模仿人的智能。神经网络则是根据人脑的生理结构和信息处理过程，来创造人工神经网络，其目的也是模仿人的智能。模仿人的智能这是它们共同的奋斗目标和合作的基础。

从知识的表达方式来看，模糊系统可以表达人的经验性知识，便于理解，而神经网络只能描述大量数据之间的复杂函数关系，难于理解；从知识的存储方式来看，模糊系统将知识存在规则集中，神经网络将知识存在权系数中，都具有分布存储的特点；从知识的运用方式来看，模糊系统和神经网络都具有并行处理的特点，模糊系统同时激活的规则不多，计算量小，而神经网络涉及的神经元很多，计算量大；从知识的获取方式来看，模糊系统的规则靠专家提供或设计，难于自动获取．而神经网络的权系数可由输入输出样本中学习，无需人来设置。因此将两者结合起来，在处理大规模的模糊应用问题方面将表现出优良的效果。

2. 10. 1 模糊神经网络主要形式

根据模糊系统(Fuzzy System，简称为 FS)和神经网络连接的形式和使用功能，两者融合可分为下列三种形式：

(1) 松散型结合

在一个系统中，对于可用“If-then”规则来表示的部分，用模糊系统描述，而对很难用“If-then”规则表示的部分，则用神经网络，两者之间没有直接联系。

(2) 并联型结合

模糊系统和神经网络在系统中按并联方式连接，即享有共同的输入。按照两系统起的作用的轻重程度，还可分为同等型和补助型，如图 2-11-1 所示。在同等型中，系统的输出由两者共同确定，而在补助型中，系统的输出主要由子系统 1(可以是 FS 或神经网络)确定，而子系统 2 的输出起补偿作用。这种情况往往是在周围环境产生变化时，子系统 1 的输出会产生偏差，需要子系统 2 的补偿。

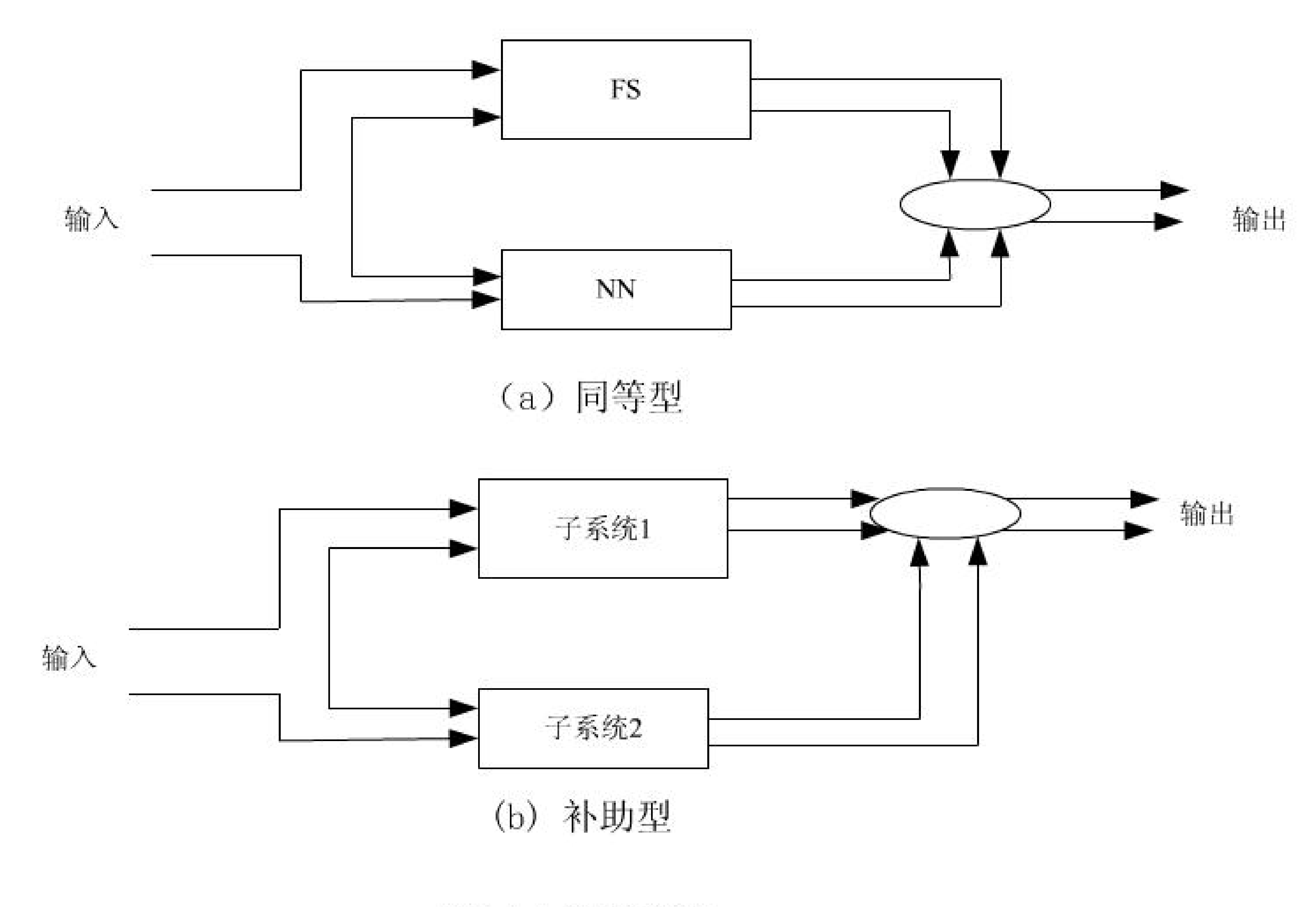


图2-24 并联模型

(3) 串联型结合

模糊系统和神经网络在系统中按串联方式连接，即一方的输出成为另一方的输入，这种情况可看成是两段推理或者串联中前者为后者输入信号的预处理部分。例如用神经网络从原输入信号提取有效的特征量，作为模糊系统的输入，这样可使获取模糊规则的过程变得容易。

2.11.2 模糊神经网络模型

基于模糊逻辑系统的标准前馈模糊神经网络结构如图2-11-2所示。

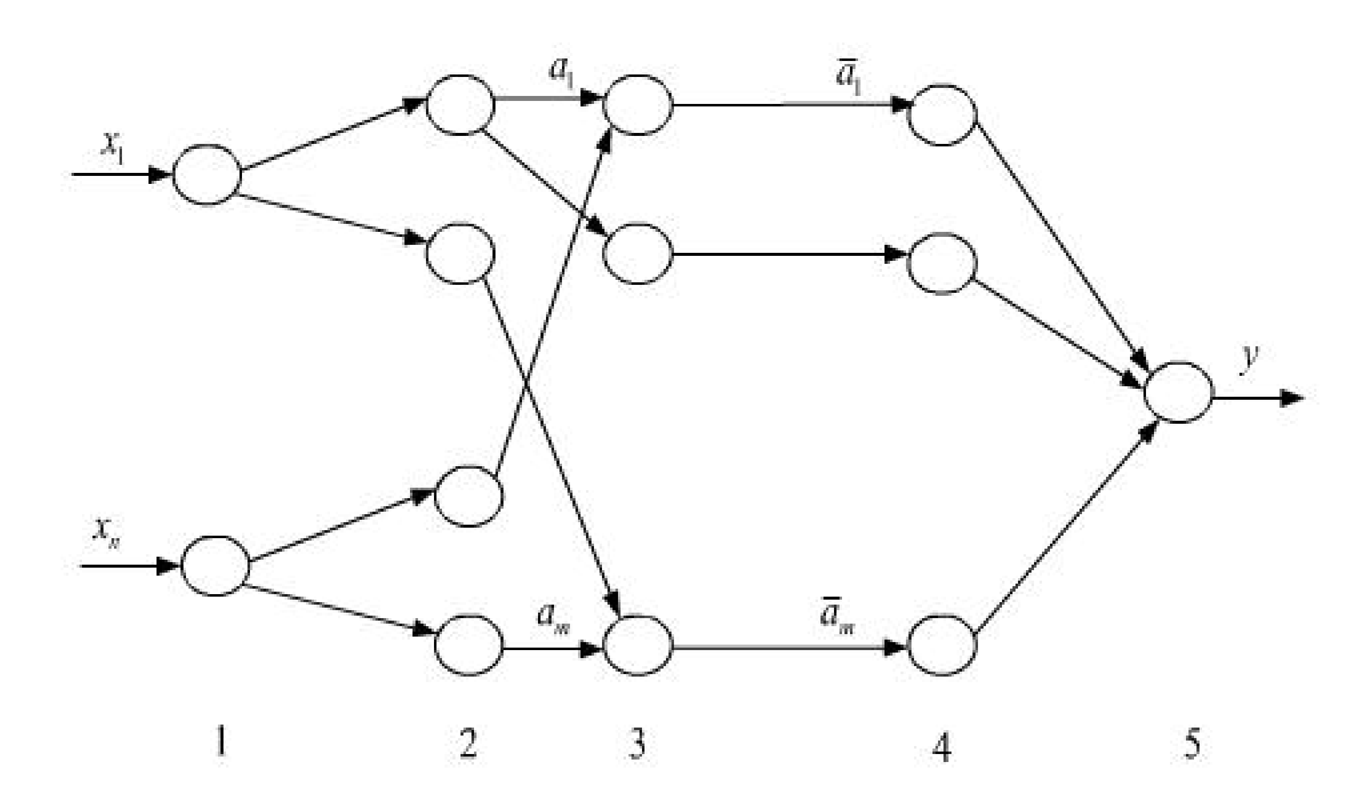


图 2-25 基于标准模型的模糊神经网络的结构

该模糊神经网络是多层网络，除了输入结点层和输出结点层外，还有多个隐含层，相邻两层之间都有连接，且每个连接都对应于一个权值。该模糊神经网络共有五层：

(1) 输入层 该层有 n 个结点直接与输入向量 $\mathbf{x}$ 连接，将输入值 $\mathbf{x}=[x_1,\cdots$ 传送到下一层。

(2) 模糊化层 若每个输入变量均定义有 m 个模糊集合，则此层内共有 n×m 个结点，分为 n 组，每组 m 个结点。第 i 组的 m 个结点输入都是 x_i，其输出分别是各输入量属于输出值模糊集合的隶属度函数 $\mu_i^j(x_i)$，$\mu_i^j(x_i)$ 代表 x_i 的第 j 个模糊集合。隶属函数通常为高斯函数。

(3) 规则层 其每个结点代表一条模糊规则，它的作用是用来匹配模糊规则的前件，计算出每条规则的使用度，即：

$$\alpha_j=\mu_1^{i1}\mu_2^{i2}\cdots$$

(4) 去模糊层 该层的作用实现归一化计算，即：

$$\overline{\alpha_j}=\frac{\alpha_j}{\sum_{i=1}^{\mathrm{m}}\alpha_i}$$

(5) 输出层 它实现的是清晰化计算，并采用加权平均法，即：

$$y=\sum_{j=1}^{m}w_j\overline{\alpha_j}$$

2.11.3 模糊神经网络学习方法

模糊学习是一种以模糊理论为基础的学习方法。这种学习方法的特点是学习过程用模糊量进行测量和计算，即输入量是经过模糊化后的模糊量。以连接权作为参数的输入与输出的映射关系也是一种模糊集合的运算关系。

常见的模糊学习算法主要有两类：一类是模糊规则的提取学习算法。模糊规则的提取指从给定的输入输出模糊数据对中发现其对应的映射关系或关联关系，这也可以认为是数据挖掘的一项任务。若给出的仅有模糊输入而无相应的模糊输出，则对输入向量一般需要进行模糊聚类或对模糊输入空间的划分。如模糊竞争学习算法即为此类学习算法，模糊规则的学习还可以是事先人为地给出一个较粗的模糊规则，然后利用学习样本反复修正这一模糊规则，进行逐步优化。另一类模糊学习算法是利用模糊控制方法不断改善神经元网络的性能，如模糊 BP 算法等，在传统的学习算法中，权值的修改是按照模型的特点和相应的规则进行的，如连续模型的梯度下降法，学习速度的控制可采用模糊规则方法进行控制，方法简单，易理解。另外，对于难以采用梯度法的离散问题的学习，模糊规则方法也能奏效。近年来，将模糊规则渗透到遗传算法中，所构成的各种模糊遗传算法已开始得到人们的重视。

2.11.4 模糊逻辑 MATLAB 函数

在 MATLAB 模糊逻辑工具箱中，提供了有关模型神经网络神经系统建模和初始化模糊推理系统的函数，读者可以将前面介绍的神经网络工具箱函数和模糊逻辑工具箱结合起来，完成自己特定情况的应用需要。同样，也可以利用 MATLAB 工具箱中的相关函数来扩展 nnToolKit 工具包，制做出满足自己需要的完全脱离 MATLAB 环境的工具。表 2-11-1 列出了模糊神经系统的重要函数和基本功能。

表 2-11-1　模糊神经系统函数

函数名	功能
newfis()	创建新的模糊推理系统
readfis()	从磁盘读出存储的模糊推理系统
getfis()	获得模糊推理系统的特征数据

writefis()	保存模糊推理系统
showfis()	显示添加注释了的模糊推理系统
setfis()	设置模糊推理系统的特性
plotfis()	图形显示模糊推理系统的输入—输出特性
addvar()	添加模糊语言变量
rmvar()	删除模糊语言变量
addrule()	向模糊推理系统添加模糊规则函数
parsrule()	解析模糊规则函数
showrule()	显示模糊规则函数
evalfis()	执行模糊推理计算函数
defuzz()	执行输出去模糊化函数
gensurf()	生成模糊推理系统的输出曲面并显示函数
anfis( )	模糊神经系统的建模函数
genfis1( )	采用网格分割方式生成模糊推理系统函数
fcm( )	模糊 C-均值聚类函数
subclust( )	减法聚类函数
genfis2( )	基于减法聚类的模糊推理系统建模函数

练习题

1.本章中各种神经网络的特点是什么？各适合于在什么情况使用？

2.查阅 MATLAB 帮助，熟悉本章中各种神经网络对应的 MATLAB 工具函数的使用方法。

3.学习率 η 在神经网络中起什么作用？如何确定它的初始值？

4.在网络上搜索本章中各种神经网络的一个应用实例。

第 3 章 神经网络优化方法

3.1 BP 网络学习算法的改进

标准 BP 算法采用的是最速梯度下降法修正权值，训练过程从某一起点沿误差函数的斜面逐渐达到最小点使之误差为零。而对于复杂的网络，误差曲面在多维空间，这就象一个碗一样，碗底就是最小点。但这个碗的表面是凹凸不平的，因而在训练过程中可能会陷入某个局部最小点，由该点向多方向变化均会使误差增加，以致无法逃出这个局部最小点。由于标准 BP 网络学习算法存在与输入样本的顺序有关、收敛速度缓慢、易陷入局部极小等缺陷，为了克服算法中的不足，下面几节将给出一些改进算法，这些改进算法或者减少了输入样本顺序的影响，或者收敛速度比标准梯度法快数十乃至数百倍。

对算法产生影响的主要是权值向量修改方法，在下面的改正算法中，仅给出了权值向量修改访求，算法的其它部分请参考 2.3.2 节中的标准 BP 算法。下面的改进算法 3.1.1~3.1.4 与标准 BP 学习算法的原理相同，均是基于梯度下降法的算法；改进算法 3.1.5~3.1.7 与它们不同，是基于数值优化的算法，这些改进算法不仅利用了误差函数的一阶导数信息，而且还利用了误差函数的二阶导数信息。在本节中，改进算法的 MATLAB 实现只需要将 2.3.3 节中的默认训练函数用本节中提到的 MATLAB 训练函数替换，即可实现改进算法，出于篇幅的原因，在此不再一一给出具体 MATLAB 代码。

3.1.1 消除样本输入顺序影响的改进算法

在 2.3.2 节描述的 BP 网络标准学习算法，在每输入一个学习样本后，根据其产生的误差立即对权值进行调整，属于在线学习方式。实验表明，在线学习方式时，网络受后面输入样本的影响较大，严重时，会影响用户要求的训练精度。为了消除这种样本顺序对结果的影响，可以采用批处理学习方式，即使用一批学习样本产生的总误差来调整权值，用公式表示如下：

$$\Delta w_{ij} = \sum \Delta_{\mathrm{p}} w_{ij}$$

式中，$\sum \Delta_{\mathrm{p}} w_{ij}$ 代表连接权 w_{ij} 关于一批学习样本的调整量，表示由所有学习样本产生的误差，每一个样本产生的权值调整量可由 2.3.2 节中相应的计算公式求得，从而有效地消除样本顺序对学习算

法的影响。仍以 2.3.2 节中的三层网络结构为例，但不考虑阈值的调整，消除样本顺序影响的 BP 改进算法流程如下：

1.网络初始化 给 $\mathbf{w}_{ih}$ 、 $\mathbf{w}_{ho}$ 分别赋一个区间（-1，1）内的随机数，设定误差函数 $e=\frac{1}{2}\sum_{o=1}^{q}(d_o(k)-y_o(k))^2$，给定计算精度值 ε 和最大学习次数 M。

2.计算输出层的权值调整值 输出层对于一批学习样本的权值调整值按下式计算：

$$\mathbf{w}_{ho}^{N+1}=\mathbf{w}_{ho}^{N}+\sum\Delta_{\mathrm{p}}w_{ho}$$

3、计算隐含层的权值调整值 。隐含层对于一批学习样本的权值调整值按下式计算：

$$\mathbf{w}_{ih}^{N+1}=\mathbf{w}_{ih}^{N}+\sum\Delta_{\mathrm{p}}w_{ih}$$

4.计算全局误差 E

$$E=\frac{1}{2m}\sum_{k=1}^{m}\sum_{o=1}^{q}(d_o(k)-y_o(k))^2$$

5.判断网络误差是否满足要求，当 $E<\varepsilon$ 或学习次数大于设定的最大次数 M，则结束算法。否则，返回到第 2 步，进入下一轮学习过程。

上述改进算法较好地解决了因样本输入顺序引起的精度问题和训练的抖动问题。但是，该算法的收敛速度相对来说还是比较慢的。

3.1.2 附加动量的改进算法

该方法是在反向传播法的基础上在每一个权值（或阈值）的变化上加上一项正比于上一次权值（或阈值）变化量的值，并根据反向传播法来产生新的权值（或阈值）变化。

带有附加动量因子的权值调节公式为：

$$\Delta\mathbf{w}(k+1)=(1-m_c)\eta\nabla\mathrm{f}(\mathbf{w}(k))+m_c(\mathbf{w}(k)-\mathbf{w}(k-1))$$

式中，$\mathbf{w}$ 为权值向量，k 为训练次数，$m_c\ (0\le m_c\le 1)$ 为动量因子，一般取 0.95 左右，η 为

学习率，$\nabla \mathrm{f}(\mathbf{w}(k))$ 为误差函数的梯度。

附加动量法的实质是将最后一次权值（或阈值）变化的影响，通过一个动量因子来传递。当动量因子取值为零时，权值（或阈值）的变化仅是根据梯度下降法产生；当动量因子取值为 1 时，新的权值（或阈值）变化则是设置为最后一次权值（或阈值）的变化，而依梯度法产生的变化部分则被忽略掉了。由此可以看出民，增加动量项后，促使权值的调节向着误差曲面底部的平均方向变化，当网络权值进入误差曲面底部的平坦区时，可以防止 $\Delta\mathbf{w}=0$ 的出现即最后一次权值的变化量为 0,，有助于使网络从误差曲面的局部极小值中跳出。但对于大多数实际应用问题，该法训练速度仍然很慢。

MATLAB 中的工具函数 traingdm()即对应于附加动量法。

3.1.3 采用自适应调整参数的改进算法

采用自适应调整参数的改进算法的基本设想是学习率 η 应根据误差变化而自适应调整，以使权系数调整向误差减小的方向变化，其迭代过程可表示为：

$$\mathbf{w}(k+1)=\mathbf{w}(k)-\eta\nabla \mathrm{f}(\mathbf{w}(k))$$

通过研究发现，在一定范围内增大学习率 η，可大大加快学习效率，得到比标准 BP 算法更快的收敛速度。然而，在 $\nabla \mathrm{f}(\mathbf{w}(k))$ 很小的情况下，采用自适应调整参数的改进算法仍然存在权值的修正量很小的问题，致使学习率降低。

MATLAB 中的工具函数 traingda()即对应于自适应调整参数法。

3.1.4 使用弹性方法的改进算法

BP 网络通常采用 Sigmoid 隐含层。当输入的函数很大时，斜率接近于零，这将导致算法中的梯度幅值很小，可能使网络权值的修正过程几乎停顿下来。弹性方法只取偏导数的符号，而不考虑偏导数的幅值。其权值修正的迭代过程可表示为：

$$\mathbf{w}(k+1)=\mathbf{w}(k)-(\mathbf{w}(k)-\mathbf{w}(k-1))sign(\nabla f(\mathbf{w}(k)))$$

式中，$sign(\cdot)$ 为符号函数。

在弹性 BP 算法中，当训练发生振荡时，权值的变化量将减小；当在几次迭代过程中权值均朝一个方向变化时，权值的变化量将增大。因此，使用弹性方法的改进算法，其收敛速度要比前几种方法快得多，而且算法并不复杂，也不需要消耗更多的内存。

3.1.5 使用拟牛顿法的改进算法

梯度法的缺点是搜索过程收敛速度较慢，牛顿法在搜索方向上比梯度法有改进，它不仅利用了准则函数在搜索点的梯度，而且还利用了它的二次导数，就是说利用了搜索点所能提供的更多信息，使搜索方向能更好地指向最优点。它的迭代方程为：

$$\Delta \mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{D}^{-1}\nabla \mathrm{f}(\mathbf{w}(k))$$ $\mathbf{D}$为$\mathrm{f}(w^k)$的二阶偏导数矩阵

牛顿法是一种常见的快速优化方法，其收敛速度比一阶梯度快。但由于牛顿法中用到 Hessian 矩阵（二阶导数矩阵），而导致计算复杂性增加。而拟牛顿法是对牛顿法进行了改进的一类无需计算二阶导数及其逆运算的方法，它通常是利用梯度信息或一个近似矩阵去逼近 Hessian 矩阵（二阶导数矩阵）。比较典型的有 BFGS 拟牛顿法和一步正切拟牛顿法。

MATLAB 中的工具函数 trainbfg()、trainoss()即对应拟牛顿法中的 BFGS 拟牛顿法和一步正切拟牛顿法。

3.1.6 基于共轭梯度法的改进算法

梯度下降法收敛速度较慢，而拟牛顿法计算又较复杂，而共轭梯度法则力图避免两者的缺点。共轭梯度法也是一种改进搜索方向的方法，它是把前一点的梯度乘以适当的系数，加到该点的梯度上，得到新的搜索方向。其迭代方程为：

$$\Delta \mathbf{w}(k+1) = \mathbf{w}(k) + \rho(k)S(k)$$

式中，$\rho(k)$是最佳步长,$S(k) = -\nabla \mathrm{f}(\mathbf{w}(k)) + \mathbf{v}(k-1)S(k-1)$

$$\mathbf{v}(k-1) = \frac{\left\|\nabla \mathrm{f}(\mathbf{w}(k))\right\|^2}{\left\|\nabla \mathrm{f}(\mathbf{w}(k-1))\right\|^2}$$

共轭梯度法比大多数常规的梯度下降法收敛快，并且只需增加很少的存储量和计算量。对于权值很多的网络，采用共轭梯度法不失为一种较好的选择。

MATLAB 中的工具函数 traincgb()、traincgf()、traincgp()即对应于共轭梯度法。

3.1.7 基于 Levenberg-Marquardt 法的改进算法

众所周知，梯度下降法在最初几步下降较快，但随着接近最优值，由于梯度趋于零，致使误差函数下降缓慢，而牛顿法则可在最优值附近产生一个理想的搜索方向。Levenberg-Marquardt 法实际上是梯度下降法和牛顿法的结合，它的优点在于网络权值数目较少时收敛非常迅速。应用 Levenberg-Marquardt 优化算法比传统的 BP 及其它改进算法（如共轭梯度法，附加动量法、自适应调整法及拟牛顿法等）迭代次数少，收敛速度快，精确度高。因此，Levenberg-Marquardt 优化算法在 BP 网络学习中具有一定优越性。在 MATLAB 神经网络仿真工具函数中，Levenberg-Marquardt 法被作为 BP 神经网络默认的训练函数，下面将给出其算法。

算法基本思想是使其每次迭代不再沿着单一的负梯度方向，而是允许误差沿着恶化的方向进行搜索，同时通过在最速梯度下降法和高斯－牛顿法之间自适应调整来优化网络权值，使网络能够有效收敛，大大提高了网络的收敛速度和泛化能力。

L-M 优化算法，又称为阻尼最小二乘法，其权值调整公式为

$$\Delta w = (\mathbf{J}^T \mathbf{J} + \mu I)^{-1} \mathbf{J}^T e \tag{3-1}$$

e 为误差向量，$\mathbf{J}$ 是误差对权值微分的雅可比矩阵，μ 是一个标量，当 μ 增加时，它接近于具有较小的学习速率的最速下降法，当 μ 下降到 0 时，该算法就变成了高斯－牛顿法了，因此，L-M 算法是在最速梯度下降法和高斯－牛顿法之间的平滑调和。

L－M 算法具体的迭代步骤为：

1）将所有输入送到网络并计算出网络的输出，另用误差函数计算出训练集中所有目标的误差平方和。

2）计算出误差对权值微分的雅可比矩阵 $\mathbf{J}$。

首先，定义 Marquardt 敏感度

$$S_i^m = \frac{\partial \mathrm{E}}{\partial n_i^m} \tag{3-2}$$

从式（3-2）可以看出，敏感度为误差函数 E 对 m 层输入的第 i 个元素变化的敏感性，其中 n 为每层网络的加权和。

敏感性的递推关系式为：

$$S_q^m = \dot{\mathrm{E}}(n_q^m)(w^{m+1})^T S_q^{m+1} \tag{3-3}$$

可见敏感性可由最后一层通过网络被反向传播到第一层，

$$S^m \to S^{m-1} \to \cdots \to S^2 \to S^1 \tag{3-4}$$

然后，用式(3-5)计算雅各比矩阵的元素，

$$[\mathbf{J}]_{h,l} = \frac{\partial \mathrm{e}_{k,q}}{\partial w_{i,j}^m} = \frac{\partial \mathrm{e}_{k,q}}{\partial n_{i,q}^m} \times \frac{\partial \mathrm{n}_{i,q}^m}{\partial w_{i,j}^m} = S_{i,h}^m \times \frac{\partial \mathrm{n}_{\mathrm{i,q}}^{\mathrm{m}}}{\partial w_{i,j}^m} = S_{i,h}^m \times a_{j,q}^{m-1} \tag{3-5}$$

3）用式(3-1)求出 Δw。

4）用 $w+\Delta w$ 重复计算误差的平方和 如果新的和小于 1）中计算的和，则用 μ 除以 θ（$\theta>1$），并有 $w=w+\Delta w$，转 1)；否则，用 μ 乘以 θ，转 3)。当误差平方和减小到某一目标误差时，算法即被认为收敛。

MATLAB 中的工具函数 trainlm()即对应 Levenberg-Marquardt 法的改进算法。

3.2 基于遗传算法的神经网络优化方法

3.2.1 概述

BP 神经网络是人工神经网络中应用最广泛的算法，但是也存在着一些缺陷：一是学习收敛速度太慢；二是不能保证收敛到全局最小点；三是网络结构不易确定。3.1 节中讲述的各种改进算法虽然能够改善 BP 网络算法的一些不足，但是在实际应用中仍然不够完善，不能完全克服 BP 算法固有的缺陷。另外网络结构、初始连接权值和阈值的选择对网络训练的影响很大，但是又无法准确获得，针对这些特点可以采用遗传算法对神经网络进行优化。遗传算法应用于神经网络的一个方面是用来优化人工神经网络(ANN)的结构，另一个方面是用遗传算法学习神经网络的权值，也就是用遗传算法取代一些传统的学习算法。

遗传算法（Genetic Algorithm，简写为 GA）是模拟达尔文的遗传选择和自然淘汰的生物进化过程的计算模型，它是由美国密执根(Michigan)大学的 J.Holland 教授于 1975 年首先提出的，遗传算法具有很强的宏观搜索能力和良好的全局优化性能，因此将遗传算法与 BP 网络相结合，训练时先用遗传算法对神经网络的权值进行寻优，将搜索范围缩小后，再利用 BP 网络来进行精确求解，可以达到全局寻优和快速高效的目的，并且可以避免局部极小问题。该算法不仅具有全局搜索能力，而

且提高了局部搜索能力，从而增强了在搜索过程中自动获取和积累搜索空间知识及自适应地控制搜索过程的能力，从而使结果的性质得以极大的改善。

3.2.2 遗传算法简介

首先将问题求解表示成基因型（如常用的二进制编码串），从中选取适应环境的个体，淘汰不好的个体，把保留下来的个体复制再生，通过交叉、变异等遗传算子产生新一染色体群。依据各种收敛条件，从新老群体中选出适应环境的个体，一代一代不断进步，最后收敛到适应环境个体上，求得问题最优解。遗传算法中概念与生物遗传学中的概念的对应关系如表 3-1 所示。

表 3-1　生物遗传学概念与遗传算法中概念的对应关系

生物遗传学概念	遗传算法中的作用
适者生存	在算法停止时，最优目标值的解有最大的可能被留住
个体（individual）	目标函数的解
染色体（chromosome）	解的编码（向量）
基因（gene）	解中的每一分量的特征（或值）
适应性（fitness）	适应函数值
群体（population）	选定的一组解（其中解的个数为群体的规模）
种群（reproduction）	根据适应函数选取的一组解
交配（crossover）	按交配原则产生一组新解的过程
变异（mutation）	编码的某一分量发生变化的过程

传统遗传算法的实现步骤如下：

1.随机产生一定数目的初始个体（染色体） 这些随机产生的染色体组成一个种群，种群中的染色体数目称为种群的规模或大小（pop-size）；

2.用评价函数来评价每个染色体的优劣　染色体对环境的适应程度（称为适应度），并用作以后遗传操作的依据；

3.基于适应值的选择策略　从当前种群中选取一定的染色体作为新一代的染色体（染色体的适应度越高，其被选择的机会越大）；

4.对这个新生成的种群进行交叉（交配）操作、变异操作（变异操作的目的使种群中的个体具

有多样性，防止陷入局部最优解），这样产生的染色体群（种群）称为后代；

对新生的种群重复进行选择、交叉、变异操作过程，经过给定次数的迭代后，把最好的染色体作为优化问题的最优解。

遗传算法的流程图如图 3-1 所示。

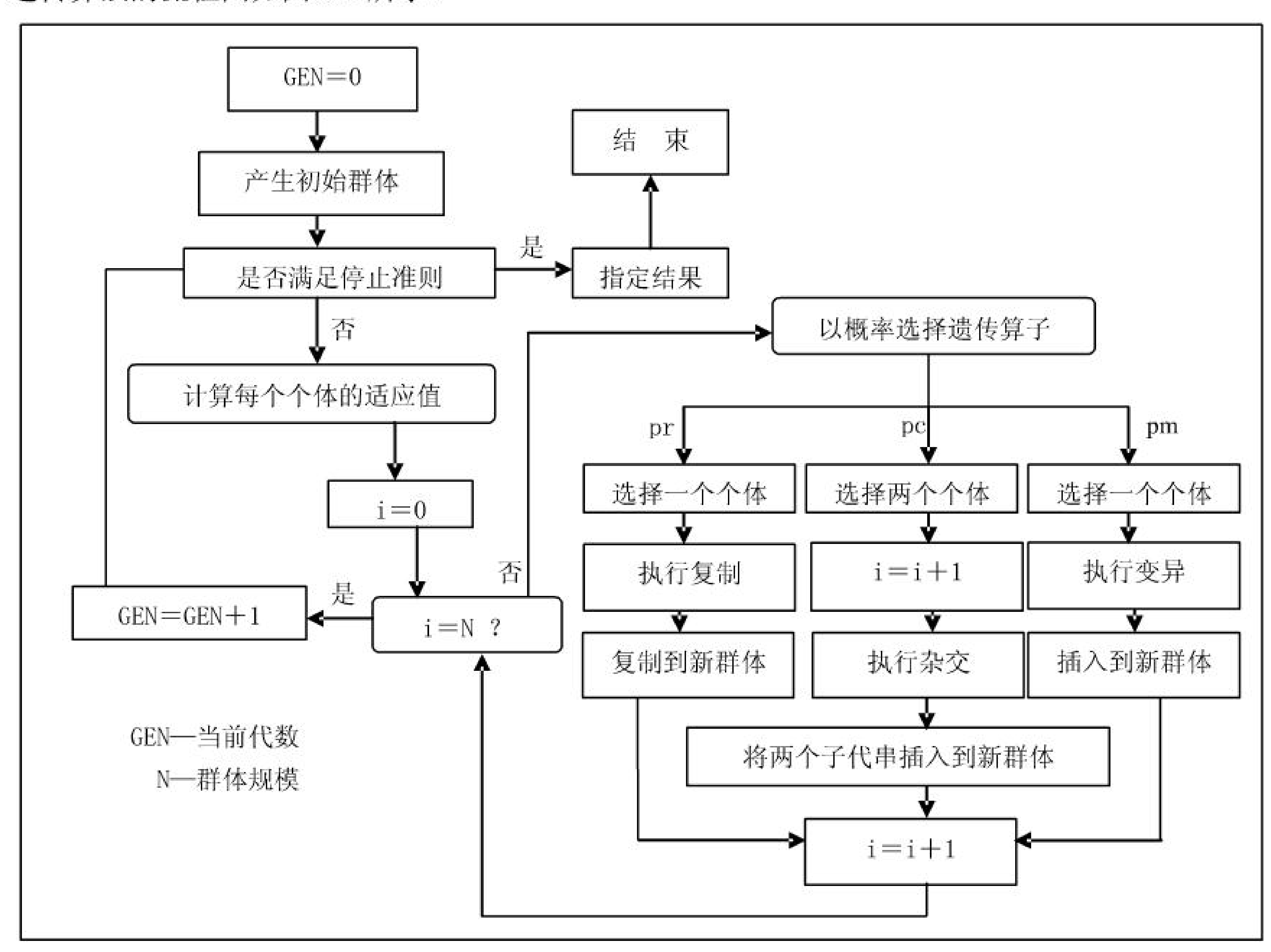


图 3-1　遗传算法流程图

3.2.3 遗传算法工具箱

MATLAB 中的遗传算法工具箱中含有丰富的遗传算法相关的函数，利用遗传算法工具箱可以很方便地实现遗传算法对神经网络权值和阈值的优化。下面给出遗传算法工具箱中几个核心函数的格式和使用方法。

1.编码和种群生成

指令格式：

```
function [pop]=initializega(populationSize, variableBounds,evalFN,
                            evalOps,options)
```

参数说明：
pop：随机生成的初始种群
populatoinSize：种群大小即种群中个体的数目
variableBounds：表示变量边界的矩阵
evalFN：适应度函数
evalOps：传给适应度函数的参数
options：选择编码形式：1 为浮点编码
0 为二进制编码
2.进行遗传操作
指令格式：

```
function [x,endPop,bPop,traceInfo]=ga(bounds,evalFN,evalOps,startPop,opts,
                                      termFN,termOps,selectFN,selectOps,
                                      xOverFNs,xOverOps,mutFNs,mutOps)
```

参数说明：
1）输出参数：
X：求得的最优解
endPop：得到的最终种群
bPop：最优种群的搜索轨迹
traceInfo：每代的最优值和均值矩阵
2）输入参数：
Bounds：代表变量上下界的矩阵
startPop：可以从初始化函数中得到的初始解矩阵
evalFN：适应度函数
termFN：终止函数的名称
termOps：终止函数的参数
selectFN：选择函数名
selectOpts：选择参数
xOverFNS：交叉函数名
xOverOps：交叉参数

mutFNs：变异函数名
mutOps：变异参数

3.2.4 用遗传算法优化神经网络权值的学习过程

仍以 2.3.2 中的三层 BP 网络为例，遗传算法学习 BP 网络的步骤如下：

1）初始化种群 P，包括交叉规模、交叉概率 Pc、突变概率 Pm 以及对任一 w_{ih} 和 w_{ho} 初始化；在编码中，采用实数进行编码，初始种群取 n (视实际应用选择数值大小)；

2）计算每一个个体评价函数，并将其排序，可按下式概率值选择网络个体：

$$p_i = f_i / \sum_{i=1}^{N} f_i$$

其中 f_i 为个体 i 的适配值，可用误差平方和 E 来衡量，即：

$$f_i = 1/E(i) \qquad E(i) = \sum_k \sum_o (d_o - yo_o)^2$$

其中 $i = 1,2,\cdots$ 为染色体数；$o = 1,2,\cdots$ 为输出层节点数；$k = 1,2,\cdots$ 为学习样本数；yo 为网络的实际输出，d 为期望输出。

3）以交叉概率 Pc 对个体 G_i 和 G_{i+1} 进行交叉操作，产生新个体 $G_i^{'}$ 和 $G_{i+1}^{'}$，没有进行交叉操作的个体直接进行复制；

4）利用变异概率 Pm 突变产生 G_j 的新个体 $G_j^{'}$；

5）将新个体插入到种群 P 中，并计算新个体的评价函数；

6）判断算法是否结束。如果找到了满意的个体，则结束，否则转 3）进入下一轮运算。

算法结束，如达到预先设定的性能指标后，将最终群体中的最优个体解码即可得到优化后的网络连接权值系数。

用遗传算法优化神经网络的 MATLAT 实现请参阅 4.5.3 节。

3.3 小波神经网络

3.3.1 概述

如果将小波变换应用到神经网络中，可以优化神经网络，从本质上改善神经网络的学习功能。小波神经网络（Wavelet Neural Network, WNN）或小波网络（Wavelet Network）是小波函数对神经网络的优化与应用，是小波分析理论与神经网络理论相结合的产物。最早研究小波分析与神经网络的联系的是 Pati 和 Krishnaprasad，他们提出了离散仿射小波网络模型，其基本思想是将离散小波变换引入神经网络模型，通过对 Sigmoid 函数的平移伸缩构成 $L^2(R)$ 中的仿射框架，进而构造小波神经网络。法国著名的信息科学研究机构 IRLSA 的 Zhang Qinghu 等 1992 年正式提出小波神经网络的概念，其思想是用小波元代替神经元，即用已定位的小波函数代替 Sigmoid 函数作为激活函数，通过仿射变换建立起小波变换与网络系数之间的连接，并应用于逼近 $L(R^n)$ 中的函数 $f(x)$。

小波神经网络是基于小波分析所构造的一种新型的神经网络模型，它充分继承了小波分析与神经网络两者的优点，因而具有更好的性能。一方面，小波变换通过尺度伸缩和平移对信号进行多尺度分析，能有效提取信号的局部信息；另一方面，神经网络具有自学习、自适应和容错性等特点，并且是一类通用函数逼近器。因此小波神经网络具有更强的逼近、容错能力。小波神经网络相比于其它前向的神经网络，它有明显的优点：首先，小波神经网络的基元和整个结构是依据小波分析理论确定的，可以避免 BP 神经网络等结构设计上的盲目性；其次，小波神经网络有更强的学习能力，精度更高；最后对同样的学习任务，小波神经网络结构更简单，收敛速度更快。

从结构形式看，小波分析与神经网络的结合有以下形式：

1．松散型结合 小波分析仅作为神经网络的前置处理手段，为神经网络提供输入特征向量，即小波分析对神经网络的输入进行初步处理，使得输入神经网络的信息更易于神经网络进行处理。

2．融合型结合 小波和神经网络直接融合，即小波元代替神经元。它是将常规神经网络的隐含层函数用小波函数来代替，相应的输入层到隐含层的权值及隐含层阈值分别由小波函数的尺度和平移参数所代替。根据基函数和学习参数的不同选取，小波神经网络又可分为如下三种形式的网络：

1)连续参数的小波神经网络 这种小波神经网络类似于径向基函数神经网络，但借助于小波分析理论，可使网络具有较简单的拓扑结构和较快的收敛速度，但由于尺度和平移参数均可调，使其与输出为非线性关系，通常需利用非线性优化方法进行参数修正，易带来类似 BP 网络参数修正时存在局部极小的弱点。

2)由框架作为基函数的小波神经网络 由于不考虑正交性，小波函数的选取有很大自由度，网络的可调参数只有权值，其与输出呈线性关系，可通过最小二乘法或其它优化法修正权值，使网络能

充分逼近目标。这种形式的网络虽然基函数选取灵活，但由于框架可以是线性相关的，使得网络函数的个数有可能存在冗余，对过于庞大的网络需考虑优化结构的算法。

3)正交基小波网络 网络隐节点由小波函数节点和尺度函数节点构成，尽管正交小波网络在理论上研究较为方便，但是正交基函数的构造复杂，不如一般的基于框架的小波网络实用。

目前对小波神经网络参数的学习算法，主要有梯度下降法、正交搜索法、矩阵求逆法等。特别地，当小波基函数的平移和伸缩也可事先确定时，小波神经网络的可调参数只有权系数，由于小波神经网络的输出与其权值是线性的，因而不存在如常规 BP 网络那样的局部极小点，可直接利用最小二乘方法。

小波神经网络虽然有很多优点，但在目前的实际应用中还存在着以下一些不足之处：

1.在多维输入情况下，随着网络的输入维数增加，网络所训练的样本呈指数增长，网络结构也将随之变的庞大，使得网络收敛速度大大下降；

2.隐含层节点数难以确定；

3.小波网络中初始化参数问题，若尺度参数与位移参数初始化不合适，将导致整个网络学习过程的不收敛；

4.未能根据实际情况来自适应选取合适的小波基函数。

3.3.2 小波神经网络参数调整算法

一般来说，小波神经网络的连接权值、尺度系统和平移系数等参数是需要通过对网络进行训练来确定的，确定小波神经网络中的各个参数有许多算法，但是最常用的仍是最速梯度下降法，因此 BP 算法的各种改进算法同样适用于小波神经网络。下面将通过详细的算法描述来说明如何确定小波神经网络中的各个待定的参数。

按照标准 BP 算法的操作流程，小波神经网络的权值的调整过程分为两个阶段：第一阶段是从网络的输入层开始逐层开始向前计算，根据输入样本计算各层的输出，最终求出网络输出层的输出，这是前向传播过程。第二个阶段是对权值的修正，从网络的输出层开始逐层向后进行计算和修正，这是反向传播过程。两个过程反复交替，直到达到要求为止。

设小波神经网络为三层网络，包括输入层、隐含层和输出层，输出层采用线性输出，输入层有 $M(m=1,2,\cdots$ 个神经元，隐含层有 $K(k=1,2,\cdots$ 个神经元，输出层有 $N(n=1,2,\cdots$ 个神经元，其结构如图 3-2 所示。隐含层选取的神经元激励函数为 Morlet 小波，Morlet 小波表达式如下：

$$\mathrm{h}(\frac{x-b}{a})=\cos(1.75\frac{x-b}{a})*\exp(-0.5(\frac{x-b}{a})^2)$$

训练时，在权值和阈值的修正算法中加入动量项，利用前一步得到的修正值来平滑学习路径，避免陷入局部极小值，加速学习速度。为了避免在逐个样本训练时，引起权值和阈值修正时发生的振荡，采用成批训练方法，将一批样本所产生的修正值累计后进行处理。对网络的输出也并不是简单的加权求和，而是先对网络隐含层小波节点的输出加权求和，再经 Sigmoid 函数变换后，得到最终的网络输出。这样做有利于处理分类问题，同时减少训练过程中发散的可能性。

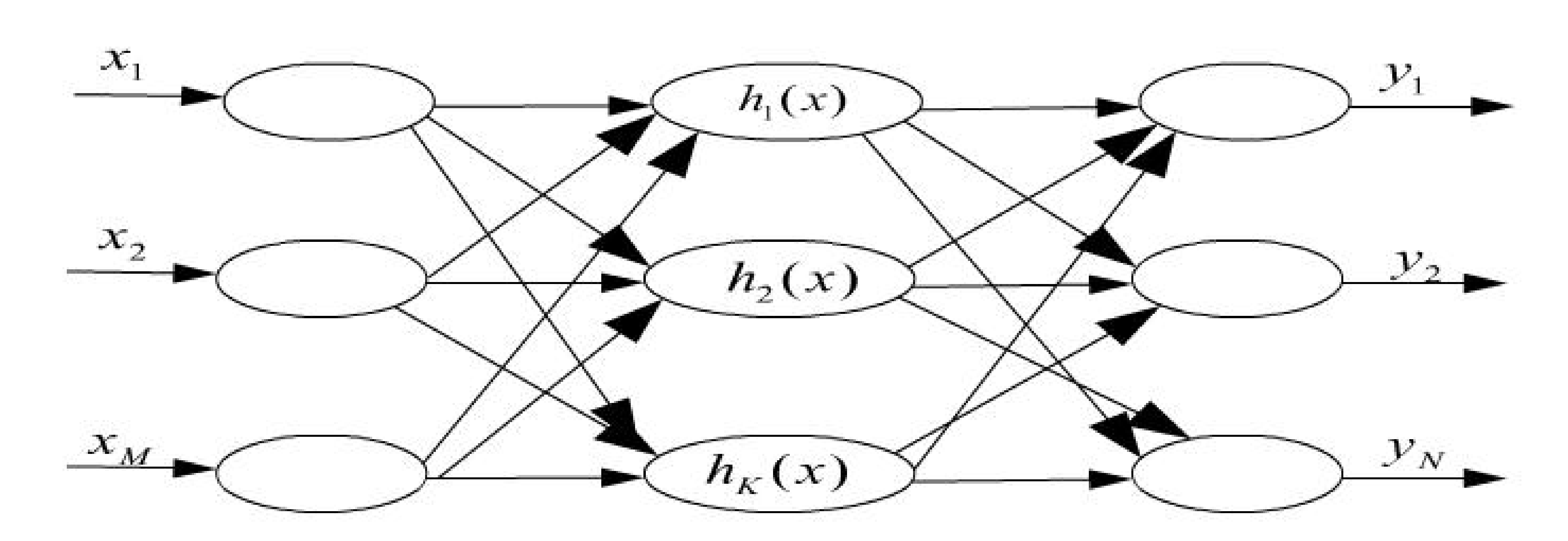


图 3-2 小波神经网络的结构

给定 $P(p=1,2,\cdots$ 组输入输出样本，学习率为 $\eta(\eta>0)$，动量因子为 $\lambda(0<\lambda<1)$，根据最速下降法的基本思想，定义如下的代价函数：x_M

$$E=\sum_{p=1}^{P}E^{p}=\frac{1}{2P}\sum_{p=1}^{P}\sum_{n=1}^{\mathrm{N}}(d_n^p-y_n^p)$$

式中，d_n^p 为输出层第 n 个节点的期望输出，y_n^p 为网络实际输出。算法的目的就是不断调整网络的各项参数，使得 E 达到最小值。

由网络结构可得隐含层输出为：

$$O_k^p=\mathrm{h}(\frac{I_k^p-b_k}{a_k}),\quad I_k^p=\sum_{m=1}^{M}w_{km}x_m^p$$

式中，x_m^p 表示输入层的输入，O_k^p 表示隐含层的输出，w_{km} 表示输入层结点 m 与隐含层结点 k 之间的权值，$\mathrm{h}()$ 表示 Morlet 小波函数。

输出层输出为：

$$y_n^p = \mathrm{f}(I_n^p), \quad I_n^p = \sum^{N} w_{nk} O_k^p$$

式中，I_n^p 表示输出层的输入，w_{nk} 表示隐含层结点 k 与输出层结点 n 之间的权值，$\mathrm{f}()$ 表示 Morlet 小波函数。

小波神经网络训练算法逐步更新神经元间的连接权值及小波函数的伸缩因子和平移因子，下面将给出它们的推导式。

隐含层与输出层之间的权值调整式：

$$w_{nk}^{new} = w_{nk}^{old} + \eta \sum_{m=1}^{p} \delta_{nk} + \lambda \Delta w_{nk}^{old}$$

式中，$\delta_{nk} = \dfrac{\partial E_n^p}{\partial w_{nk}} = (d_n^p - y_n^p) \cdot y_n^p \cdot (1 - y_n^p)$，$w_{nk}^{old}$ 和 w_{nk}^{new} 分别表示调整前与调整后的隐含层结点 k 与输出层结点 n 之间的连接权值，Δw_{nk}^{old} 表示动量项。

$$w_{km}^{new} = w_{km}^{old} + \eta \sum_{m=1}^{p} \delta_{km} + \lambda \Delta w_{km}^{old}$$

式中，$\delta_{km} = \dfrac{\partial E_n^p}{\partial w_{km}} = \sum_{n=1}^{N} (\delta_{nk} w_{nk}) \cdot \dfrac{\partial O_k^p}{\partial I_k^p} \cdot x_m^p$，$w_{km}^{old}$ 和 w_{km}^{new} 表示调整前与调整后的输入层结点 m 与隐含层结点 k 之间的权值，Δw_{km}^{old} 表示动量项。

$$a_k^{new} = a_k^{old} + \eta \sum_{m=1}^{p} \delta_{a_k} + \lambda \Delta a_k^{old}$$

式中，$\delta_{a_k} = \dfrac{\partial E_n^p}{\partial a_k} = \sum_{n=1}^{N} (\delta_{nk} w_{nk}) \cdot \dfrac{\partial O_k^p}{\partial a_k}$，$a_k^{old}$ 和 a_k^{new} 表示调整前与调整后的伸缩因子，Δa_k^{old} 表示伸缩因子动量项。

$$b_k^{new} = b_k^{old} + \eta\sum_{m=1}^{p}\delta_{b_k} + \lambda\Delta b_k^{old}$$

式中，$\delta_{b_k} = \dfrac{\partial E_n^p}{\partial b_k} = \sum_{n=1}^{N}(\delta_{nk}w_{nk})\cdot\dfrac{\partial O_k^p}{\partial b_k}$，$b_k^{old}$ 和 b_k^{new} 表示调整前与调整后的平移因子，Δb_k^{old} 表示平移因子动量项。

学习算法的具体实现步骤如下：

（1）网络参数的初始化 将小波的伸缩因子 a_k、平移因子 b_k、网络连接权值 w_{km} 和 w_{nk}、学习率 $\eta(\eta > 0)$ 以及动量因子 $\lambda(0 < \lambda < 1)$ 赋予初始值，并置输入样本计数器 $p = 1$。

（2）输入学习样本及相应的期望输出 d_n^p。

（3）计算隐含层及输出层的输出。

（4）计算误差和梯度向量。

（5）输入下一个样本，即 $p = p + 1$。

（6）判断算法是否结束。当 $E < \varepsilon$ 时，即代价函数 E 小于预先设定的某个精度值 $\varepsilon(\varepsilon > 0)$，停止网络的学习，否则将 p 重置为 1，并转步骤（2）。

3.3.3 小波神经网络的 MATLAB 函数

利用 MATLAB 中的小波神经网络工具箱函数,可以很方便地实现小波神经网络的建立和仿真，下面具体介绍静态非线性回归小波神经网络的创建过程来具体说明如何使用 MATLAB 中的小波网络工具箱函数来设计小波神经网络。

1. 静态非线性回归小波神经网络的创建

指令格式：

```
THETA = wnetreg(y, x, nbwavelon, max_epoch, initmode, min_nbw, levels)
```

参数说明：

1）输出参数

THETA 小波回归模型的估计参数。

2）输入参数

y 一个列向量。

x 对于单输入，x 是一个列向量；对于多输入，x=[x1 x2 ... xm]，每个 xi 都是一个列向量。

nbwavelon：构建小波网络的小波数量

max_epoch：最大训练次数。

initmode：初始化模式，缺省值为 2。

如果 initmode=THETA 则为一个包含小波神经网络参数的矩阵，这些参数用来初始化网络。这种模式对重复训练小波神经网络具有很大的作用。

min_nbw：最小输入模式数。

levels：初始化过程中的级别数。

min_nbw 和 levels 是可以选择的。

2. 小波神经网络的仿真

小波神经网络的仿真使用函数 wavenet( )，可以很方便地得到网络的仿真结果。

指令格式：

g = wavenet(x, THETA)

小波神经网络的 MATLAB 实现请参阅 4.5.4 节。

练习题

1、BP 网络的改进算法哪些？简要说明其原理。

2、简要说明 L-M 优化算法、遗传算法优化神经网络权值算法、小波神经网络算法的学习过程。

3、针对例子 2-3 所描述的药品的销售预测实例，现构建一个如下的三层 BP 神经网络：输入层有三个结点，隐含层节点数为 5，隐含层的激活函数为 tansig(双曲正切 S 型传递函数)；输出层结点数为 1 个，输出层的激活函数为 logsig(S 型的对数函数)，例子 2-3 已经实现了标准 BP 算法的预测过程，参考该预测过程，请用 3.1 节中描述的改进 BP 算法分别实现预测过程，并比较在相同的精度要求下各种方法所得到的训练误差和训练次数。

第 4 章 nnToolKit 神经网络工具包

4.1 nnToolKit 简介

nnToolKit 神经网络工具包是基于 MATLAB 神经网络工具箱自行开发的一组神经网络算法函数库，其中的函数在 MATLAB 环境下均可独立运行，并可打包成 DLL 组件，这些组件作为独立的 COM 对象，可以直接被 Visual Basic、Visual C++ 、C++ Builder 或其它支持 COM 的高级语言所引用。本工具包中包含的算法包括 BP 算法中的常用改进算法、模糊神经网络、小波神经网络、遗传算法优化神经网络权值改进算法等，根据需要，读者还可以对该工具包进行扩展，以实现一些特殊的算法需求。本章将对其工具包中的主要函数，以及如何扩展工具包做详细介绍。

4.2 nnToolKit 函数库

目前 nnToolKit 神经网络工具包中包含了神经网络算法的一些优化或改进算法。基于此，用户可以快速开发出基于优化神经网络算法的应用程序，同时还可以增加新的函数，对工具包进行扩展，以适应实际应用的需求。表 4-1 列出了 nnToolKit 神经网络工具包中所包含的主要函数。

表 4-1 nnToolKit 的重要函数和基本功能

算法	函数名	功　能
LM 神经网络算法	LmTrain()	LM 神经网络训练函数
	LmSimu()	LM 神经网络仿真函数
模糊神经网络	FnnTrain()	模糊神经网络训练函数
	FnnSimu()	模糊神经网络仿真函数
遗传算法优化小波神经	wnninit ()	小波神经网络初始化函数
	wnn ()	直接用小波神经网络逼近非线性(内部调用小波函数)

网络	gawnn ()	遗传算法优化小波神经网络后逼近非线性(内部调用遗传算法的初始化、适应度、解码函数)
	wnndemo()	基于小波神经网络的 1-D 插值示例程序
遗传算法优化网络权值	initnet()	根据指定的权值阈值，获得设置好的一个神经网络
	gadecod ()	将遗传算法的编码分解为 BP 网络所对应的权值、阈值
	Gafitness ()	遗传算法的适应值计算
	Generatesample ()	在指定路径生成适合于训练的样本
	getWBbyga ()	用遗传算法获取神经网络权值阈值参数
	Gabptrain ()	结合遗传算法的神经网络训练
	segment ()	利用训练好的神经网络进行分割图像
	compbpandgabp ()	传统 BP 和遗传 BP 训练示例程序
	demo ()	基于遗传神经网络的图像分割示例程序

下面对工具包中的主要函数作简要介绍。

4.2.1 LmTrain ( )

功能 LM 神经网络训练函数。

格式 retstr = LmTrain(ModelNo,NetPara,TrainPara,InputFun,OutputFun,DataDir)。

说明 函数返回网络训练次数，同时将网络训练结果(权、阈值)及训练误差保存到文件。各参数说明如下：

1) ModelNo 神经网络模型编号。
2) NetPara 神经网络参数，它是一个四维数组，分别表示输入层节点数、输出层节点数、中间层节点数和训练样本组数。
3) TrainPara 神经网络可选训练参数，当采用默认值时，参数设置为-1，它是八维数组，分别表示显示间隔次数；最大循环次数；目标误差；设置最小梯度；设定 μ 的初始值；设定 μ 的增加系数；设定 μ 的减少系数；设定 μ 的最大值。
4) InputFun 输入层到中间层的传递函数，默认值为'tansig'，当采用默认值时，参数指定为'-1'。
5) OutputFun 中间层到输出层的传递函数，默认值为'purelin'，当采用默认值时，参数指定为

'-1'。

6) DataDir 数据文件保存路径。

4.2.2 LmSimu（）

功能 LM 神经网络仿真函数。

格式 retstr = LmSimu(ModelNo,NetPara,SimulatePara,InputFun,OutputFun,DataDir)。

说明 函数调用经 LmTrain 函数训练的神经网络权阈值参数文件，对未知的输入样本进行仿真，返回仿真结果，同时将仿真结果写入结果文件。各参数说明如下：

1) ModelNo 神经网络模型编号。
2) NetPara 神经网络参数，它是一个三维数组，分别表示输入层节点数、输出层节点数和中间层节点数。
3) SimulatePara 神经网络仿真输入参数，是经归一化后的测试样本数据，其维数与神经网络输入参数个数相同，如神经网络输入参数为 7，则 SimulatePara 为一个 7 维的向量。
4) InputFun 输入层到中间层的传递函数，默认值为'tansig'，当采用默认值时，参数指定为'-1'。
5) OutputFun 中间层到输出层的传递函数，默认值为'purelin'，当采用默认值时，参数指定为'-1'。
6) DataDir 数据文件保存路径。

L-M 神经网络用到的数据文件包括：

①专家样本归一化数据文件 input_para.txt 和 output_para.txt。

input_para.txt 为归一化后的专家样本的输入参数文件，一组样本放在同一行。

output_para.txt 为归一化后的专家样本的输出参数文件，一组样本放在一行，使用时注意输入和输出的样本个数要求一致，即两个数据文件中数据行数要相同。

②保存神经网络结构的权域值文件 w1.dat；w2.dat；b1.dat；b2.dat。

由于任意一个多层的神经网络结构，都可以通过调整隐含层单元个数，由一个三层的神经网络结构来实现。故提供的组件中只包含三层的神经网络，用户可以通过调整隐含层单元个数，来实现复杂的网络结构需要。

w1.dat；b1.dat 代表输入层到中间层的权域值文件；

w2.dat；b2.dat 代表中层到输出层的权域值文件。

③神经网络训练误差文件 lm_err.dat，记录神经网络循环训练一次对应的误差。

④神经网络仿真后的结果文件 result.dat。

4.2.3 FnnTrain（）

功能 模糊神经网络训练函数。
格式 retstr = FnnTrain(dt,ld,tt,sp)。
说明 对指定的样本数据进行训练，同时将训练结果写入权值文件，各参数说明如下：
1) dt 输入参数，学习阈值。
2) ld 输入参数，学习进度。
3) tt 输入参数，训练次数。
4) sp 输入参数，样本数据文件。

4.2.4 FnnSimu（）

功能 模糊神经网络仿真函数。
格式 retstr = FnnSimu(kd,sj,td)。
说明 调用训练好的模糊神经网络模型，对输入样本进行仿真，各参数说明如下：
1) kd 输入参数，学习阈值。
2) sj 输入参数，学习进度。
3) td 输入参数，仿真输入数据。

4.2.5 initnet（）

功能 根据指定的权值阈值，获得设置好的一个神经网络。
格式 [net] = initnet(W1, B1, W2, B2,paraments)。
参数说明：
1) W1 输入参数，输入层到隐含层权值。
2) B1 输入参数，输入层到隐含层阈值。
3) W2 输入参数，隐含层到输出层权值。
4) B2 输入参数，隐含层到输出层阈值。
5) paraments 神经网络参数信息：[最大迭代次数　最小误差]。

4.2.6 gadecod（）

功能 将遗传算法的编码分解为 BP 网络所对应的权值、阈值。
格式 [W1, B1, W2, B2, P, T, A1, A2, SE, val]=gadecod(x)。

参数说明：
1) x 输入参数，一个染色体。
2) W1 输出参数，输入层到隐含层权值。
3) B1 输出参数，输入层到隐含层阈值。
4) W2 输出参数，隐含层到输出层权值。
5) B2 输出参数，隐含层到输出层阈值。
6) P 输出参数，训练样本。
7) T 输出参数，样本输出值。
8) A1 输出参数，输入层到隐含层误差。
9) A2 输出参数，隐含层到输出层误差。
10) SE 输出参数，误差平方和。
11) val 输出参数，遗传算法的适应值。

4.2.7 gafitness ( )

功能 遗传算法的适应值计算。
格式 f = gafitness(y)。
参数说明：
1) y 输入参数，染色体个体。
2) f 输出参数，染色体适应度。

4.2.8 generatesample( )

功能 在指定路径生成适合于训练的样本。
格式 [ ] = generatesample(path)。
参数说明：
1) path 输入参数，指定路径，用于保存样本文件。

4.2.9 getWBbyga ( )

功能 用遗传算法获取神经网络权值阈值参数。
格式 [W1, B1, W2, B2] = getWBbyga(paraments)。
参数说明：
1) paraments 输入参数，遗传算法的参数信息：[遗传代数 最小适应值]。

4.2.10 segment（）

功能 利用训练好的神经网络进行分割图像。

格式 [bw] = segment(net,img)。

参数说明：

1) net 输入参数，已经训练好的神经网络。
2) img 输入参数，要分割的图像。
3) bw 输出参数，分割后的二值图像。

4.2.11 gabptrain（）

功能 结合遗传算法的神经网络训练。

格式 [net] = gabptrain(gaP,bpP,P,T)。

参数说明：

1) gaP 输入参数，遗传算法的参数信息：[遗传代数 最小适应值]。
2) bpP 输入参数，神经网络参数信息：[最大迭代次数 最小误差]。
3) P 输入参数，样本数组。
4) T 输入参数，目标数组。
5) net 输出参数，训练好的网络结构。

4.2.12 compbpandgabp（）

功能 传统 BP 和遗传 BP 训练示例程序。

格式 retstr = compbpandgabp()。

说明 对于指定的样本数据，分别用传统 BP 算法和遗传 BP 算法进行训练，相比之下，用传统 BP 训练，可能收敛不到目标值，或者收敛步数太长，而用遗传 BP 算法进行训练，遗传算法寻找最优权值阈值会用一些时间，但比 BP 的训练还是快，在很短的时间内就能收敛到目标值。

4.2.13 demo（）

功能 基于遗传神经网络的图像分割示例程序。

格式 retstr = demo()。

说明 演示对特定类型的一类图像进行分割。在分割之前要做好二项工作，一是提取前景和背景的特征值，二是用提取好的特征值进行遗传神经网络训练。

4.3 程序解析

下面以 L-M 算法中的神经网络训练和仿真函数为例，予以详细说明。

1、L-M 神经网络训练函数

```
function retstr = LmTrain(ModelNo,NetPara,TrainPara,InputFun,OutputFun,DataDir)
NNTWARN OFF
retstr=-1;
%
% 输入参数赋值开始，这部分代码主要是方便用户调试用
%
%ModelNo='1';
%NetPara(1)=7;
%NetPara(2)=1;
%NetPara(3)=28;
%NetPara(4)=10;
%TrainPara(1)=25;
%TrainPara(2)=1000;
%TrainPara(3)=0.001;
%TrainPara(4)=0.001;
%TrainPara(5)=0.001;
%TrainPara(6)=10;
%TrainPara(7)=0.1;
%TrainPara(8)=1e10;
%DataDir='.';
%InputFun = 'tansig';
%OutputFun = 'purelin';
%
% 输入参数赋值结束
%

%保留原目录
olddir=pwd;
%进入数据所在目录
```

```
cd(DataDir);

deltalin(1);
deltalog(1);
deltatan(1);
% 网络参数
InputDim=NetPara(1);          %输入层节点数
OutputDim=NetPara(2);         %输出层节点数
MidDim=NetPara(3);            %中间层节点数
data_num=NetPara(4);          %训练数据组数
% 网络训练参数
if (TrainPara == -1)
    df = 25;                  %显示间隔次数 25
    me = 1000;                %最大循环次数 1000
    eg = 0.001;               %目标误差 0.001
    lr = 0.001;               %学习速率 0.001
    lr_inc = 0.001;           %学习速率增加比率 0.001
    lr_idec = 10;             %学习速率减少比率 10
    mom_const = 0.1;          %动量常数 0.1
    err_ratio = 1e10;         %最大误差比率 1e10
else
    df=TrainPara(1);          %显示间隔次数 25
    me=TrainPara(2);          %最大循环次数 1000
    eg=TrainPara(3);          %目标误差 0.02

    lr=TrainPara(4);          %学习速率 0.001
    lr_inc=TrainPara(5);      %学习速率增加比率 0.001
    lr_idec=TrainPara(6);     %学习速率减少比率 10
    mom_const=TrainPara(7);   %动量常数 0.1
    err_ratio=TrainPara(8);   %最大误差比率 1e10
end
% 输入层到中间层的传递函数
if (length(InputFun)==0)
    InputFun = 'tansig';
end
```

```
% 中间层到输出层的传递函数
if (length(OutputFun)==0)
	OutputFun = 'purelin';
end
tp=[df me eg lr lr_inc lr_idec mom_const err_ratio];
%tp=[df me eg 0.001 0.001 10 0.1 1e10];

frin_para=fopen(sprintf('input_para%s%s',ModelNo,'.txt'),'r');        %输入数据文件
frout_para=fopen(sprintf('output_para%s%s',ModelNo,'.txt'),'r');      %输出数据文件
[p,count]=fscanf(frin_para,'%f',[InputDim,data_num]);      %取输入数据
[t,count]=fscanf(frout_para,'%f',[OutputDim,data_num]); %取输出数据
fclose(frin_para);
fclose(frout_para);

[r,q]=size(p); [s2,q]=size(t);
[w1,b1]=rands(MidDim,r);
[w2,b2]=rands(s2,MidDim);

NNTWARN OFF
[w1,b1,w2,b2,epochs,errors]=trainlm(w1,b1,InputFun,w2,b2,OutputFun,p,t,tp);

%将网络训练结果写入文件
fww1=fopen(sprintf('w%s%s',ModelNo,'1.dat'),'w');
fwb1=fopen(sprintf('b%s%s',ModelNo,'1.dat'),'w');
fww2=fopen(sprintf('w%s%s',ModelNo,'2.dat'),'w');
fwb2=fopen(sprintf('b%s%s',ModelNo,'2.dat'),'w');

fprintf(fww1,'%9.4f ',w1);
fprintf(fwb1,'%9.4f ',b1);
fprintf(fww2,'%9.4f\n',w2);
fprintf(fwb2,'%9.4f\n',b2);

fclose(fww1);
fclose(fwb1);
fclose(fww2);
```

```
fclose(fwb2);

%将训练过程误差写入误差文件
ferr=fopen(sprintf('lm_err%s%s',ModelNo,'.dat'),'w');
fprintf(ferr,'%10.6f\n',errors);
fclose(ferr);
cd(olddir);
retstr=epochs;
close all;
```

函数调试：

将“输入参数赋值开始”和“输入参数赋值结束”部分的代码行注释去掉；

点击菜单“Debug/Run”或直接按 F5 键，程序即可运行，界面如图 4-1：

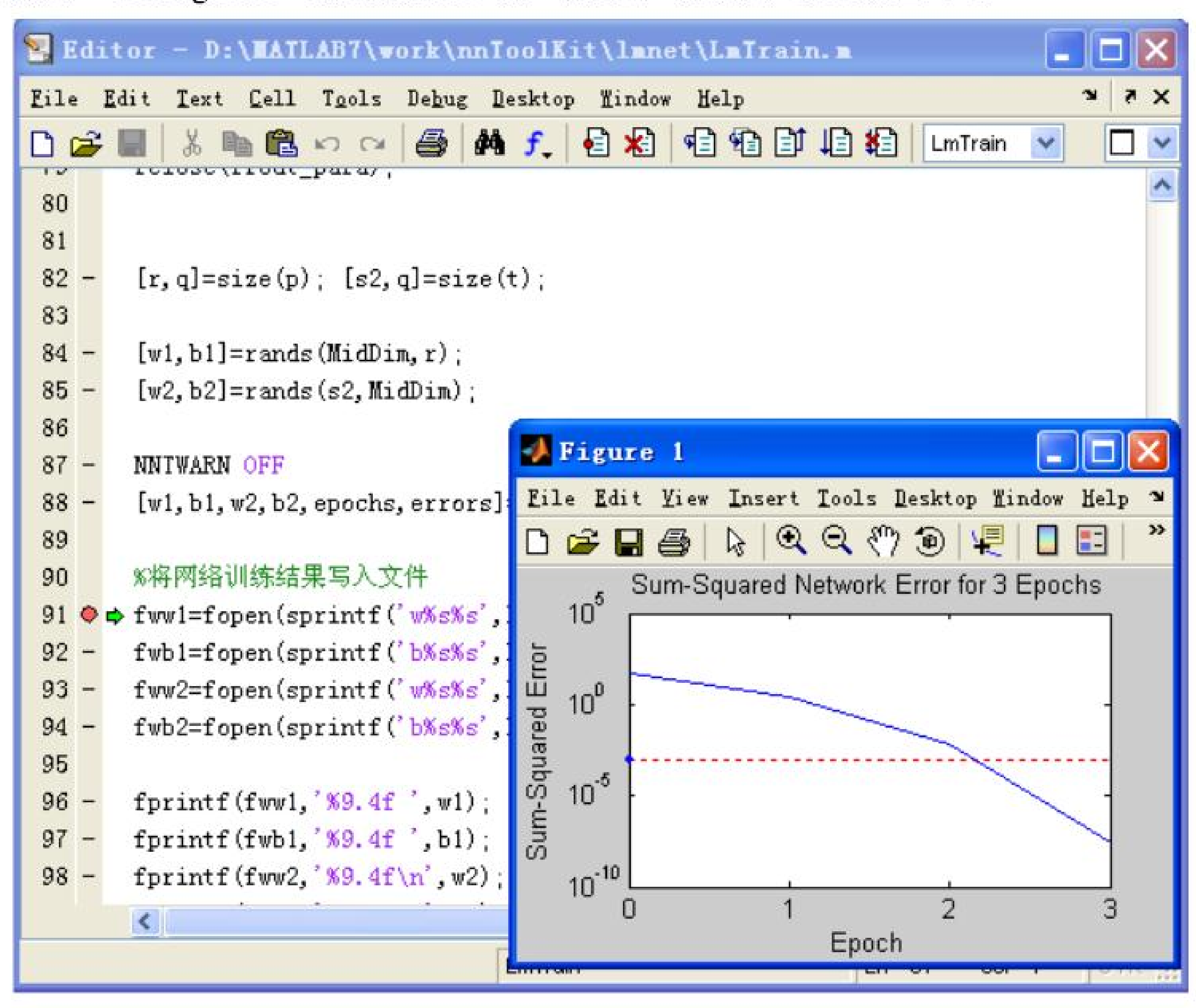


图 4-1 LM 网络训练函数调试及训练误差

程序运行完成后，会在程序的同一个文件夹中产生 b11.dat、b12.dat、w11.dat、w12.dat、lm_err1.dat 等文件。

2、L-M 神经网络仿真函数

```
function retdouble = LmSimu(ModelNo,NetPara,SimulatePara,InputFun,OutputFun,DataDir)
NNTWARN OFF
%
% 输入参数赋值开始，这部分代码主要是方便用户调试用
% ModelNo='1';
%NetPara(1)=7;
%NetPara(2)=1;
%NetPara(3)=28;

%SimulatePara(1)=0.495723;
%SimulatePara(2)=0.286143;
%SimulatePara(3)=0.849300;
%SimulatePara(4)=0.839320;
%SimulatePara(5)=0.932467;
%SimulatePara(6)=0.428714;
%SimulatePara(7)=0.100800;

%DataDir='.';
%InputFun = 'tansig';
%OutputFun = 'purelin';
%
% 输入参数赋值结束
%
retdouble = -1;
olddir=pwd;
% 数据所在目录
cd(DataDir);

% 输入层到中间层的传递函数
if (length(InputFun)==0)
```

```
        InputFun = 'tansig';
end
% 中间层到输出层的传递函数
if (length(OutputFun)==0)
        OutputFun = 'purelin'
end

%读入训练好的神经网络的权域值参数
frw1=fopen(sprintf('w%s%s',ModelNo,'1.dat'),'r');
frb1=fopen(sprintf('b%s%s',ModelNo,'1.dat'),'r');
frw2=fopen(sprintf('w%s%s',ModelNo,'2.dat'),'r');
frb2=fopen(sprintf('b%s%s',ModelNo,'2.dat'),'r');

InputDim=NetPara(1);           %输入层节点数
OutputDim=NetPara(2);          %输出层节点数
MidDim=NetPara(3);             %中间层节点数

[w1,count]=fscanf(frw1,'%f',[MidDim,InputDim]);
[b1,count]=fscanf(frb1,'%f',[MidDim,1]);
[w2,count]=fscanf(frw2,'%f',[OutputDim,MidDim]);
[b2,count]=fscanf(frb2,'%f',[OutputDim,1]);

fclose(frw1);
fclose(frb1);
fclose(frw2);
fclose(frb2);

for i=1:InputDim
        p1(i)=SimulatePara(i);
end

%网络仿真
a=simuff(p1',w1,b1,InputFun,w2,b2,OutputFun);

%将仿真结果写入文件
```

```
fwresult=fopen(sprintf('result%s%s',ModelNo,'.dat'),'w');
fprintf(fwresult,'%-15.6f\n',a);
fclose(fwresult);
retdouble = a;
logsig(1);
purelin(1);
tansig(1);
cd(olddir);
```

函数调试：

1）将“输入参数赋值开始”和“输入参数赋值结束”部分的代码行注释去掉；

2）点击菜单“Debug/Run”或直接按 F5 键，程序即可运行，运行完成后，会在函数同一文件夹下产生 result.dat 网络测试结果文件。

4.4 工具包扩展

针对实际需要，有时用户希望能对 nnToolKit 神经网络工具包进行扩展，增加一些自定义函数，这一过程是非常简单的，步骤如下：

1．仿照前一节，编写用户自已的函数（.m）。

2．打开工程文件 nnToolKit.cbl，将自定义函数加入到工程中。

3．编译，打包。

4.5 应用举例

4.5.1 基于 LM 神经网络的房地产开发风险预测模型

1．案例描述

房地产开发存在风险，其影响因素（输入）主要包括：

通货膨胀风险，可分为升高、不变、降低。

1)　市场低供求风险，可分为供大求、平衡、供小求。

2) 周期风险，可分为第一阶段、第二阶段、第三阶段、第四阶段。

3) 利率风险，可分为提高、不变、降低。

4) 政策风险，可分为有利、无新政策、不利开发。

5) 区位风险，可分为升值、不变、降值。

6) 开发期风险，可分为较长、正常、缩短。

开发风险（输出）可分为 5 类：无风险、风险较低、一般风险、风险较高、风险很高，为了能对开发风险进行有效地评估和预测，在此拟建立神经网络预测模型，实现对房地产开发风险的预测。

2．程序实现

根据案例描述，设计如表 4-2 所示的数据映射关系，并进而模拟产生一组专家样本数据，建立一个网络结构为 7-5-1 的神经网络模型。

表 4-2 网络节点数据映射关系

分析风险类	输入	转换		输出	转换
1、通货膨胀风险	升高	0.8		无风险	1
	不变	0.5			
	降低	0.2			
2、市场低供求风险	供大于求	0.3		风险较低	0.8
	平衡	0.6			
	供小于求	0.8			
3、周期风险	第一阶段	0.1		一般风险	0.5
	第二阶段	0.3			
	第三阶段	0.6			
	第四阶段	0.8			
4、利率风险	提高	0.2		风险较高	0.3
	不变	0.5			
	降低	0.9			
5、政策风险	有利	0.9		风险很高	0.1

	无新政策	0.8			
	不利开发	0.2			
6、区位风险	升值	0.9			
	不变	0.6			
	降值	0.3			
7、开发期风险	较长	0.2			
	正常	0.5			
	缩短	0.7			

将输入和输出样本数据分别写入 input_para.txt 和 output_para.txt 文件，在 MATLAB 环境下，直接运行 LmTrain.m 文件，即可完成对指定样本数据的训练，图 4-2 显示训练过程均方误差。

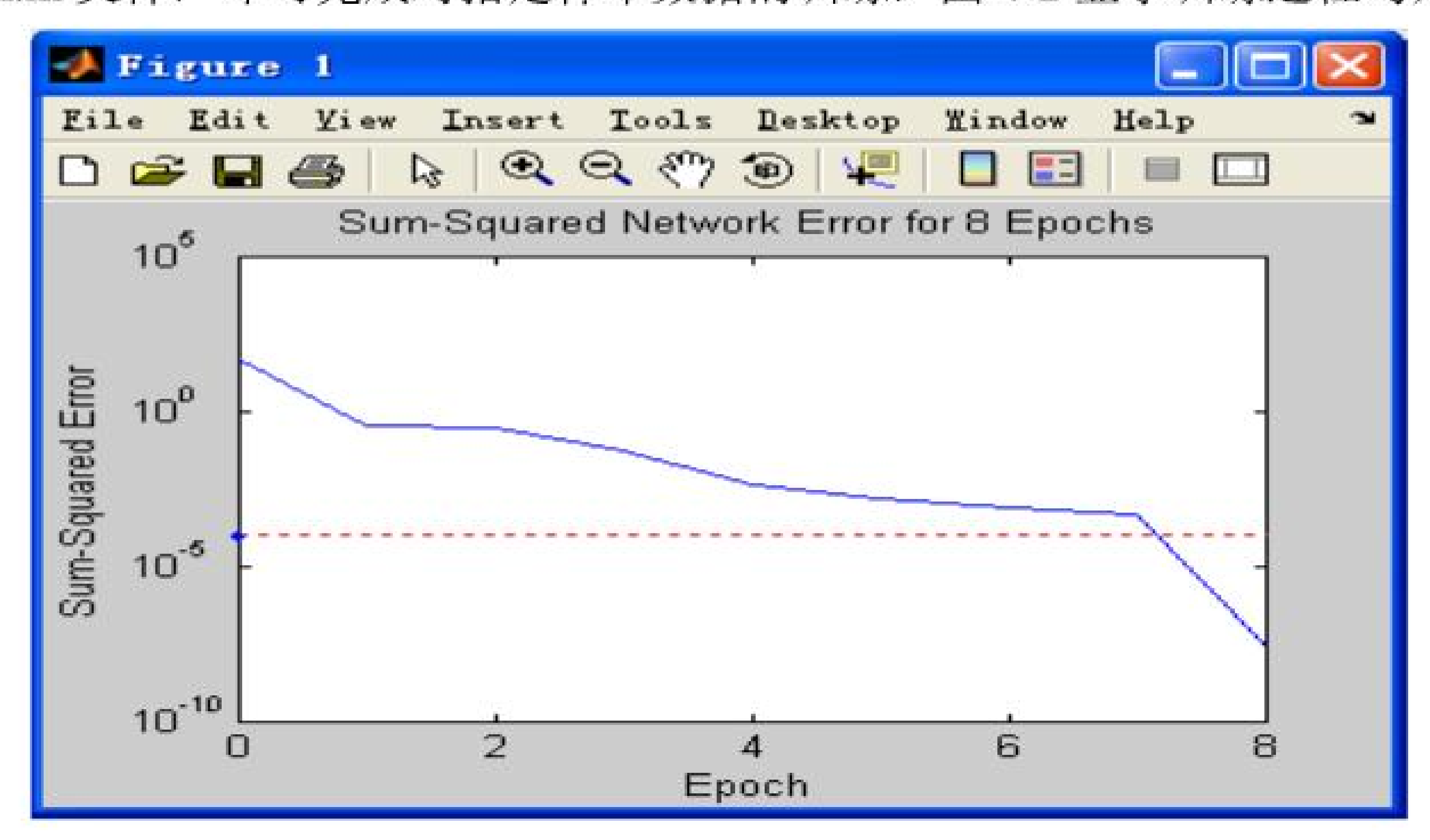


图 4-2 训练过程均方误差

要实现开发风险预测，只需将相关参数传入 LmSimu.m 文件，在 MATLAB 环境下运行即可得到预测结果。

4.5.2 模糊神经网络预测地基沉降量

1．案例描述

建筑物在施工过程中和施工结束后，往往会产生一定的沉降量，而影响结果的主要因素，又主

要与时间、土的性质、与施工面的距离、支护结构参数、工程进度快慢、地下水渗水情况等因素有关。为了能有效地预测下一个阶段（如 10 天）将要发生的沉降值或最终沉降量，这里介绍采用模糊神经网络算法，实现预测过程。经综合分析，归纳出如下几个方面作为影响预测结果（累计沉降量）的影响因素：

⑴ 时间

⑵ 距掌子面距离

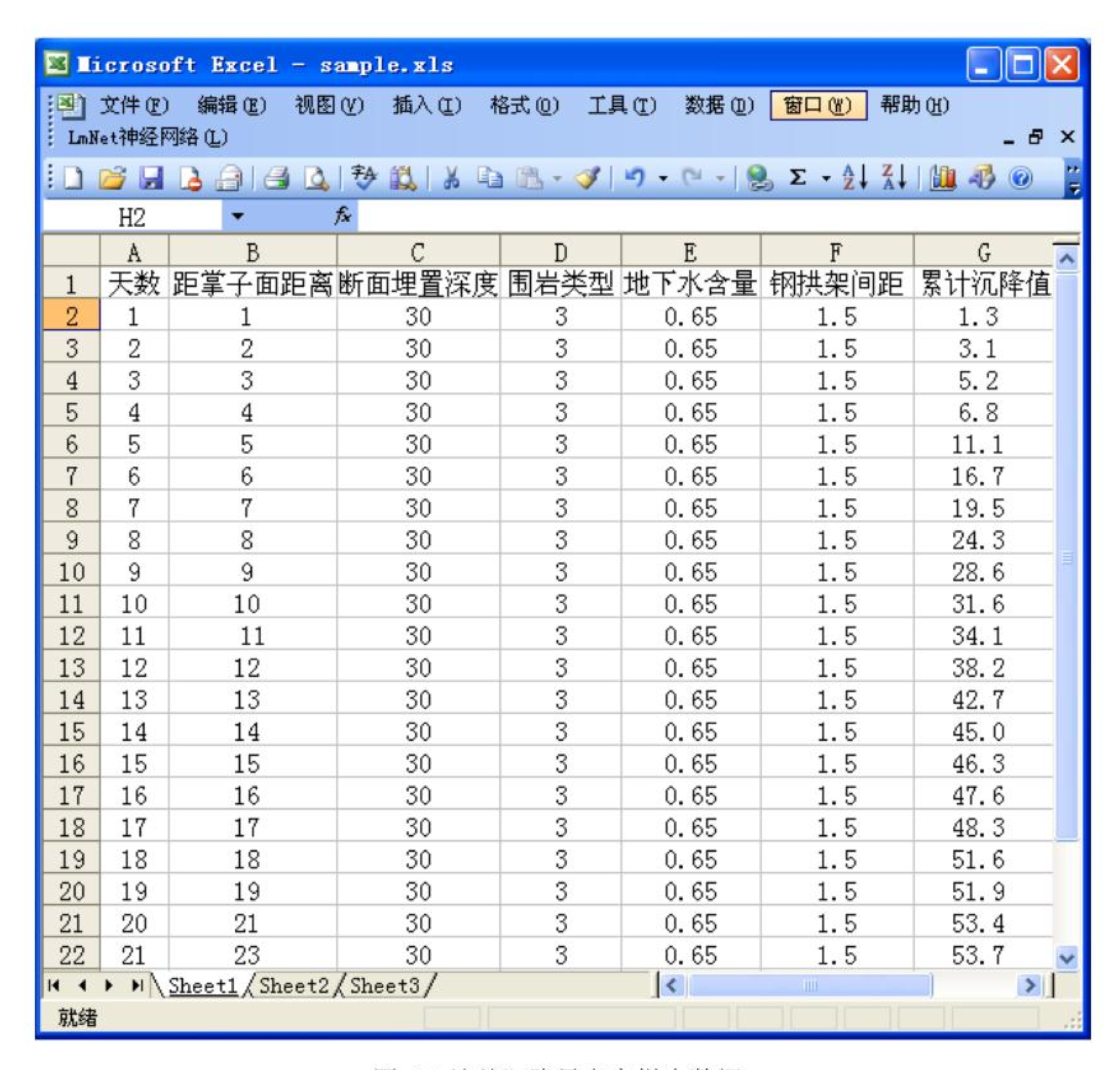
⑶ 断面埋置深度

⑷ 围岩类型

⑸ 地下水含量

⑹ 钢拱架间距

同时准备如图4-3所示的专家样本数据：



天数	距掌子面距离	断面埋置深度	围岩类型	地下水含量	钢拱架间距	累计沉降值
1	1	30	3	0.65	1.5	1.3
2	2	30	3	0.65	1.5	3.1
3	3	30	3	0.65	1.5	5.2
4	4	30	3	0.65	1.5	6.8
5	5	30	3	0.65	1.5	11.1
6	6	30	3	0.65	1.5	16.7
7	7	30	3	0.65	1.5	19.5
8	8	30	3	0.65	1.5	24.3
9	9	30	3	0.65	1.5	28.6
10	10	30	3	0.65	1.5	31.6
11	11	30	3	0.65	1.5	34.1
12	12	30	3	0.65	1.5	38.2
13	13	30	3	0.65	1.5	42.7
14	14	30	3	0.65	1.5	45.0
15	15	30	3	0.65	1.5	46.3
16	16	30	3	0.65	1.5	47.6
17	17	30	3	0.65	1.5	48.3
18	18	30	3	0.65	1.5	51.6
19	19	30	3	0.65	1.5	51.9
20	21	30	3	0.65	1.5	53.4
21	23	30	3	0.65	1.5	53.7

图 4-3 地基沉降量专家样本数据

2．代码实现

(1) 网络训练

```
function    retstr = FnnTrain(dt,ld,tt,sp)
   retstr=-1;
%
```

```
% 输入参数赋值开始，这部分代码主要是方便用户调试用
% 调试时将其中的注释去掉，直接点击菜单“Debug/Run”或直接按 F5 键，程序即可运行
%
% dt=4;            %学习阈值
% ld=0.05;         %学习进度
% tt=10;           %训练次数
% sp='data\sample.txt';      %一个点的监测数据
%
% 输入参数赋值结束
%
  global recordDimention;
  global sampleNumber;
  global weightNumber;
  global distanceThread;
  global WW;
  global learningDegree;
  global epochsNumber;

  distanceThread=dt;
  learningDegree=ld;
  traintimes=tt;
  A=load(sp);

  [Arow Acol]=size(A);
  %样本个数
  sampleNumber=Arow;
  recordDimention=Acol;
  disp(sampleNumber);
  WW=A(1,:);
  WW=[WW [1]];
  weightNumber=1;
  epochsNumber=1;

  for jj=2:1:sampleNumber
      TrainNN2(A(jj,:));
```

```
    end
    for jt=1:traintimes-1
        for jt2=1:sampleNumber
            TrainNN2(A(jj,:));
        end
    end
   % 将训练结果写入权值文件
    dlmwrite('data\w.dat',WW,'\t');
%
% 训练子函数
%
function TrainNN2(a)
    global recordDimention;
    global sampleNumber;
    global weightNumber;
    global distanceThread;
    global WW;
    global learningDegree;
    global epochsNumber;

    Ldistance=zeros(2,weightNumber);
    for j1=1:weightNumber
        Ldistance(2,j1)=j1;
    end
    % %%%%%输入输出空间的模糊分割
    for j2=1:weightNumber
        Lx=0;
        for j3=1:recordDimention
            Lx=Lx + (a(j3) - WW(j2,j3)) .* (a(j3) - WW(j2,j3));
        end
        Ldistance(1,j2)=sqrt(Lx);
    end
    % %%%%%计算模糊空间的距离
    Lx1=Ldistance(1,1);
    Lx2=Ldistance(2,1);
```

```
    for j4=2:weightNumber
        if(Ldistance(1,j4)<Lx1)
            Lx1=Ldistance(1,j4);
            Lx2=Ldistance(2,j4);
        end
    end
   % %%%%%修正模糊规则
    updateW=0;
    nowWeight=Lx2;
    if(Lx1<=distanceThread)
        for j6=1:recordDimention
            WW(nowWeight,j6)=WW(nowWeight,j6)     +     learningDegree     .*     (a(j6)     -
WW(nowWeight,j6));
        end
        WW(nowWeight,recordDimention+1)=WW(nowWeight,recordDimention+1)+1;
        updateW=1;
    end

    if( updateW==0)
        weightNumber=weightNumber+1;
        a=[a [1]];
        WW=[WW;a];
    end
```

(2) 沉降量预测

```
 function retstr = FnnSimu(kd,sj,td)
%
% 输入参数赋值开始，这部分代码主要是方便用户调试用
% 调试时将其中的注释去掉，直接点击菜单“Debug/Run”或直接按 F5 键，程序即可运行
%
% kd=4;
% sj=0.05;
% td=[37,26,15,3,0.16,1];
%
% 输入参数赋值结束
```

```
%
   kuand=kd;
   sjsd=sj;
% 加载训练好的权值文件
   WW=load('data\w.dat');
   [row,col]=size(WW);
   mu=zeros(1,row);
   b0=WW(:,col-1);
% %%%%%根据模糊规则计算预测结果
   for j=1:row
         x1=1;
         for i=1:col-2
               x1=x1 .* 1/(sjsd .* exp(((td(i)-WW(j,i))/kuand) .* ((td(i)-WW(j,i))/kuand)));
         end
         mu(j)=x1;
   end
   x2=0;
   x3=0;
   for i1=1:row
         x2=x2 + b0(i1) .* mu(i1);
         x3=x3 +mu(i1);
   end
% 返回网络预测结果：某天的累计沉降量。
retstr=x2/x3;
```

3．程序运行

在 MATLAB 命令窗口中输入 FnnTrain(4,0.05,10,'data\sample.txt')即可学习（其中 sample.txt 为某个点的监测数据），输入 FnnSimu(4,0.05,[37,26,15,3,0.16,1]) 即可得到预测结果，其中传入参数详见函数说明。另外，也可参见函数中的注释说明，在调试环境直接运行程序。

4.5.3 基于遗传神经网络的图像分割

1．案例描述

遗传神经网络在特征分类方面，有着非常广泛的应用。通过先期的学习，能够分类出特定的对

像或特征。将其用在图像分割上面，主要对特定类型的一类图像进行分割。如，在一副含有苹果和草莓的图像当中，分割出苹果或者草莓；在一副含有人的图像中，分割出肤色区域。在本例中，主要将其用在医学图像的分割上。

在分割之前要做好两项工作：一是提取前景和背景的特征值；二是用提取好的特征值进行遗传神经网络训练。

（1）　提取特征值

主要通过手工对图像进行分析，确定前景色的范围和背景色的范围，然后把前景色和背景色按顺序存入一个数组中，生成的这个数组就为训练样本数组。然后再建立一个同样大小的数组，来保存样本的特征值。如果为前景则特征值为 1，背景色特征值为 0。

（2）　遗传神经网络训练

将上一步提取好的样本值和特征值送入遗传神经网络进行训练。遗传神经网络首先在权值阈值的值空间中，搜索出一组最合适的权值和阈值，将此设置为神经网络的初始权值阈值。然后再进行训练，直到均方误差收敛到指定值，或者达到最大迭代次数。此时的神经网络是最优的。

（3）　图像分割

可以将图像分割看成一个分类的过程。图像（G）中的每一个像素（Gij）是一个待分类的样本，将这个样本送入遗传神经网络（sim）进行分类，将输出一个特征值 Vi，这个特征值决定该样本属于其中一类的概率。可以决定，如果该值大于 0.5，那么认为它是前景（F），否则它就是背景（B）。

$$Vij = sim(Gij)$$

$$Hij = \begin{cases} F & Vij >= 0.5 \\ B & Vij < 0.5 \end{cases}$$

其中，H 为分割后的图像。

2．代码实现

```
function retstr = demo()
NNTWARN OFF
retstr=-1;
%
%用于产生样本文件
generatesample('data\sample.mat');
%
%遗传神经网络训练示例
gaP = [100 0.00001];
bpP = [500 0.00001];
load('data\sample.mat');
gabptrain( gaP,bpP,p,t )
```

```
%
%神经网络分割示例
load('data\net.mat');
img = imread('image\a.bmp');
bw = segment( net,img ) ;
figure;
subplot(2,1,1);
imshow(img);
subplot(2,1,2);
imshow(bw);
%
%传统 BP 训练
%出现的结果，可能收敛不到目标值，或者收敛步数太长(356 步)
epochs = 2000;
goal = 0.00001 ;
net = newcf([0 255],[6 1],{'tansig' 'purelin'});
net.trainParam.epochs = epochs;
net.trainParam.goal = goal ;
load('data\sample.mat');
net = train(net,p,t);
%
%遗传 BP 训练
%遗传算法寻找最优权值阈值会用一些时间，
%bp 的训练还是非常快，38 步就收敛到的目标值
gaP = [100 0.00001];
bpP = [500 0.00001];
gabptrain( gaP,bpP,p,t );
```

3．程序运行

程序在 MATLAB7.0 下运行，要求安装有遗传算法工具包，直接运行 demo.m 文件。主要界面分别如图 4-4~4-7 所示：

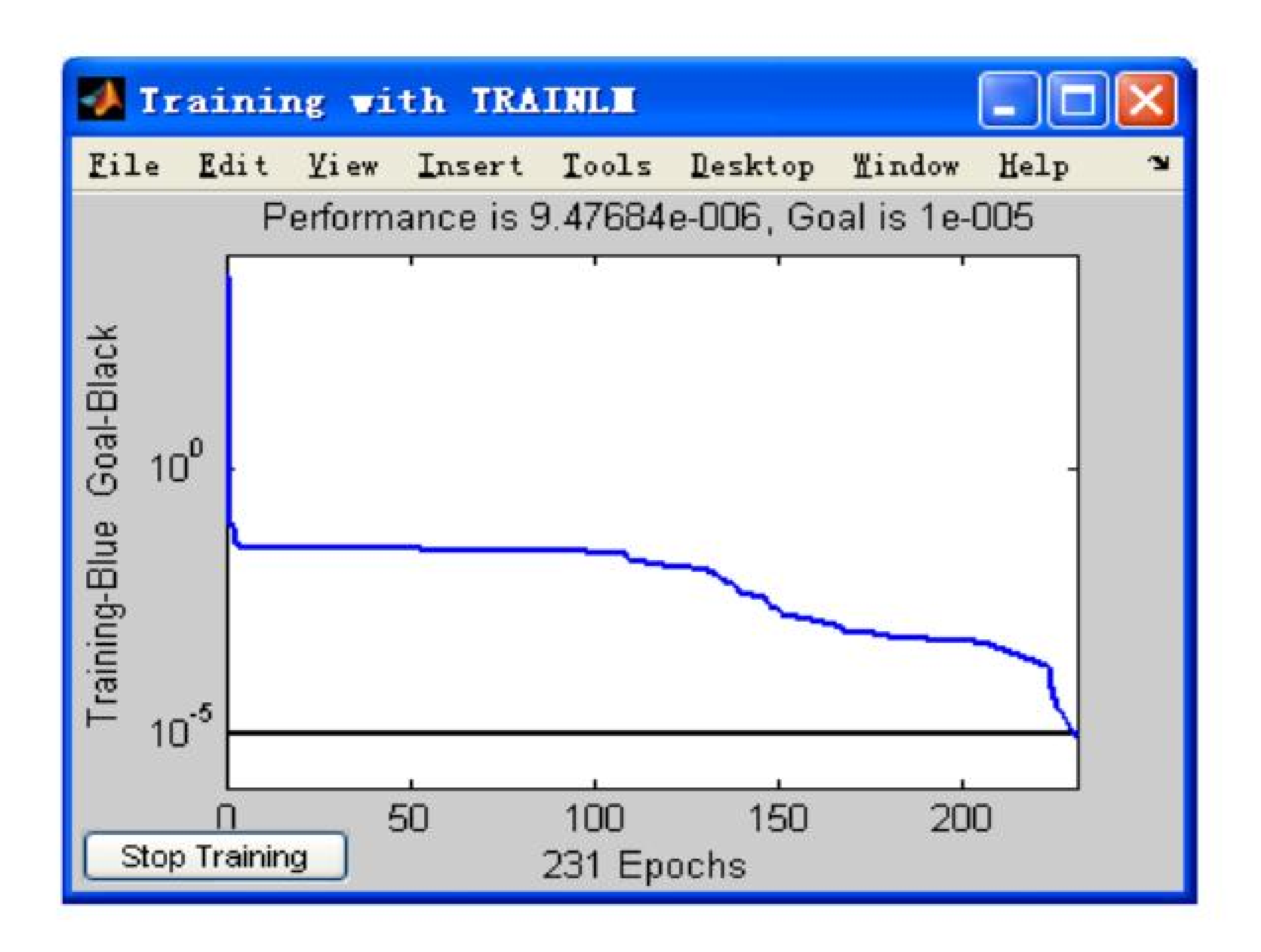


图 4-4 结合遗传算法的神经网络训练过程误差

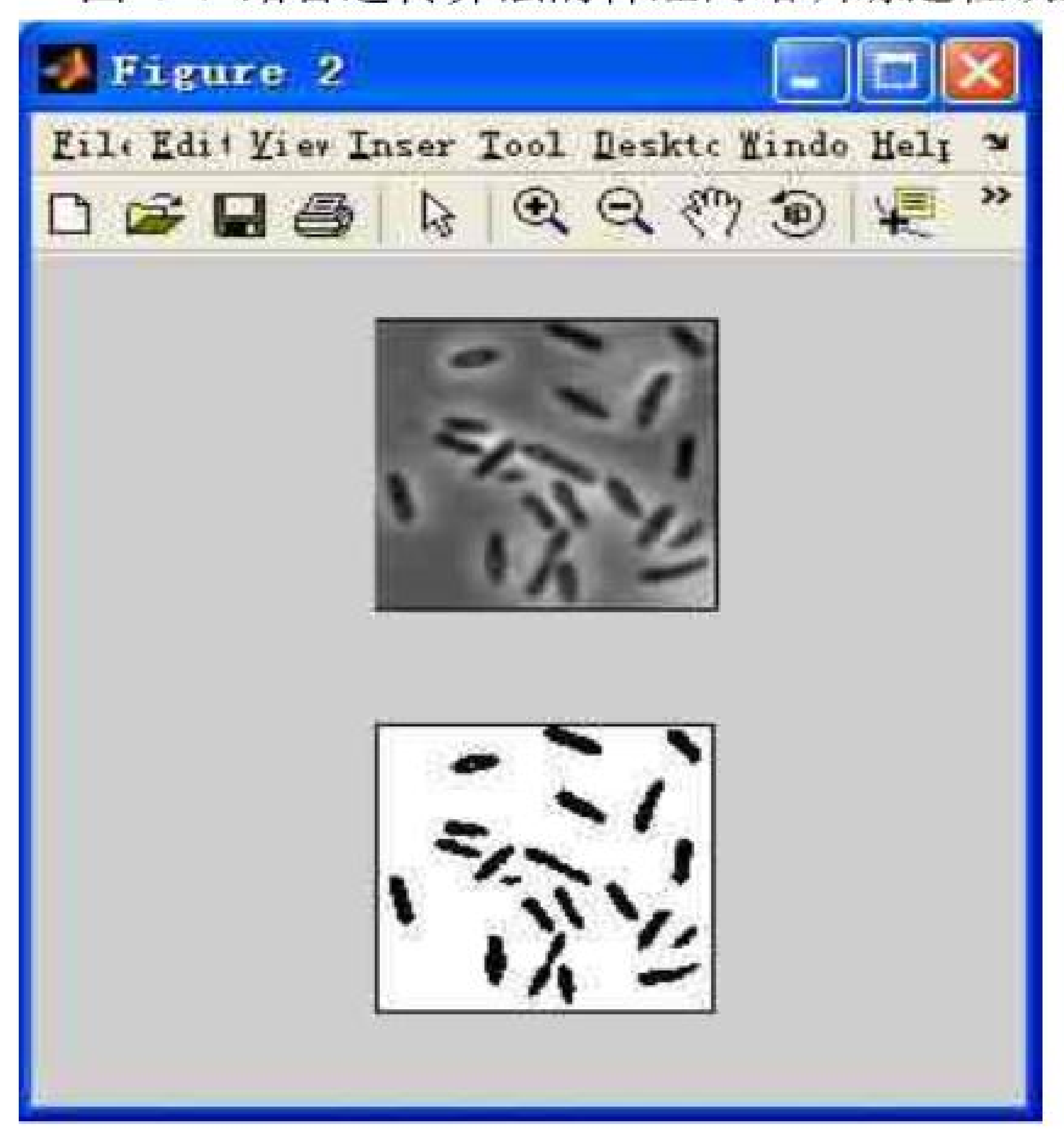


图 4-5 图像分割结果

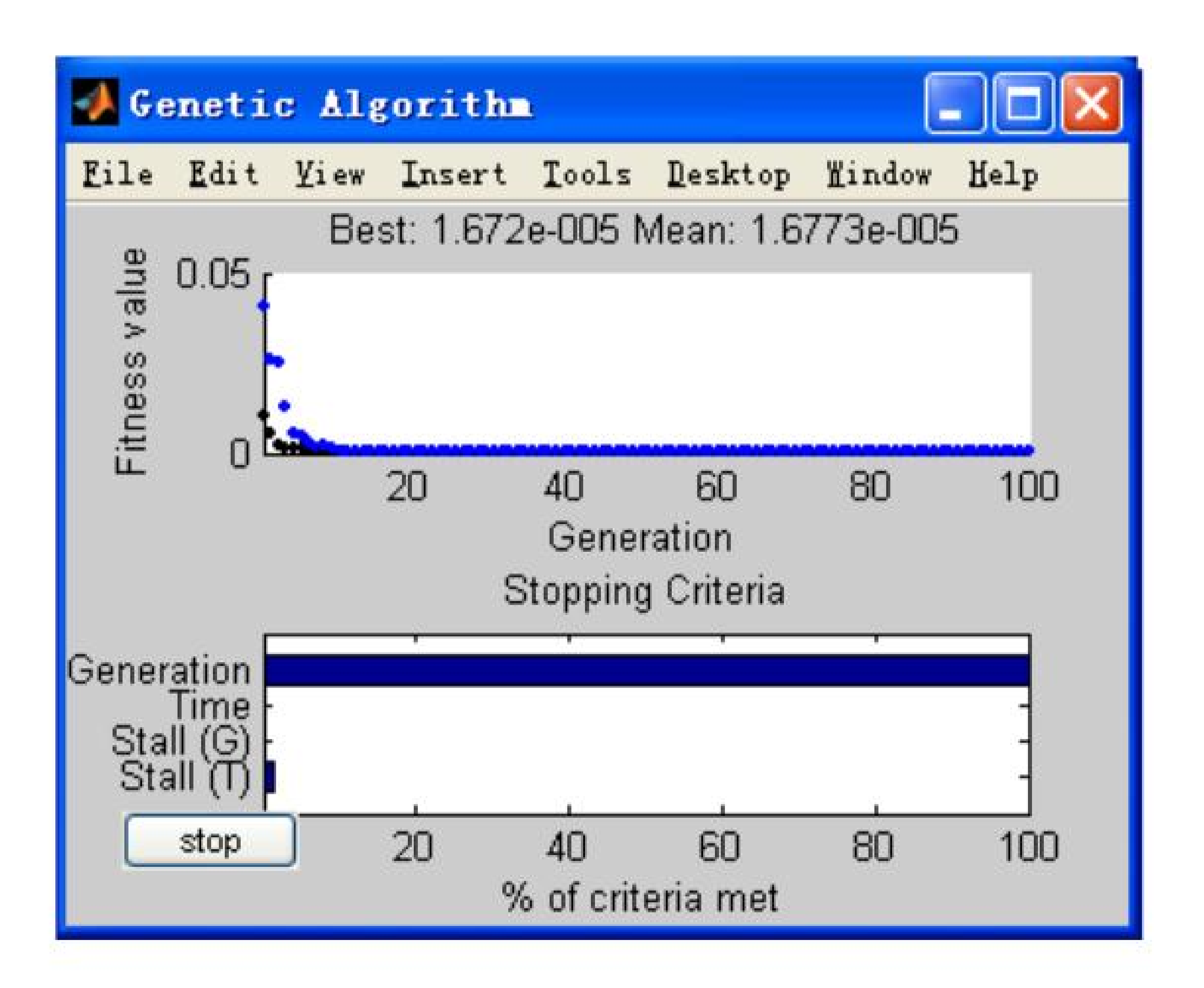


图 4-6 遗传算法适应值计算

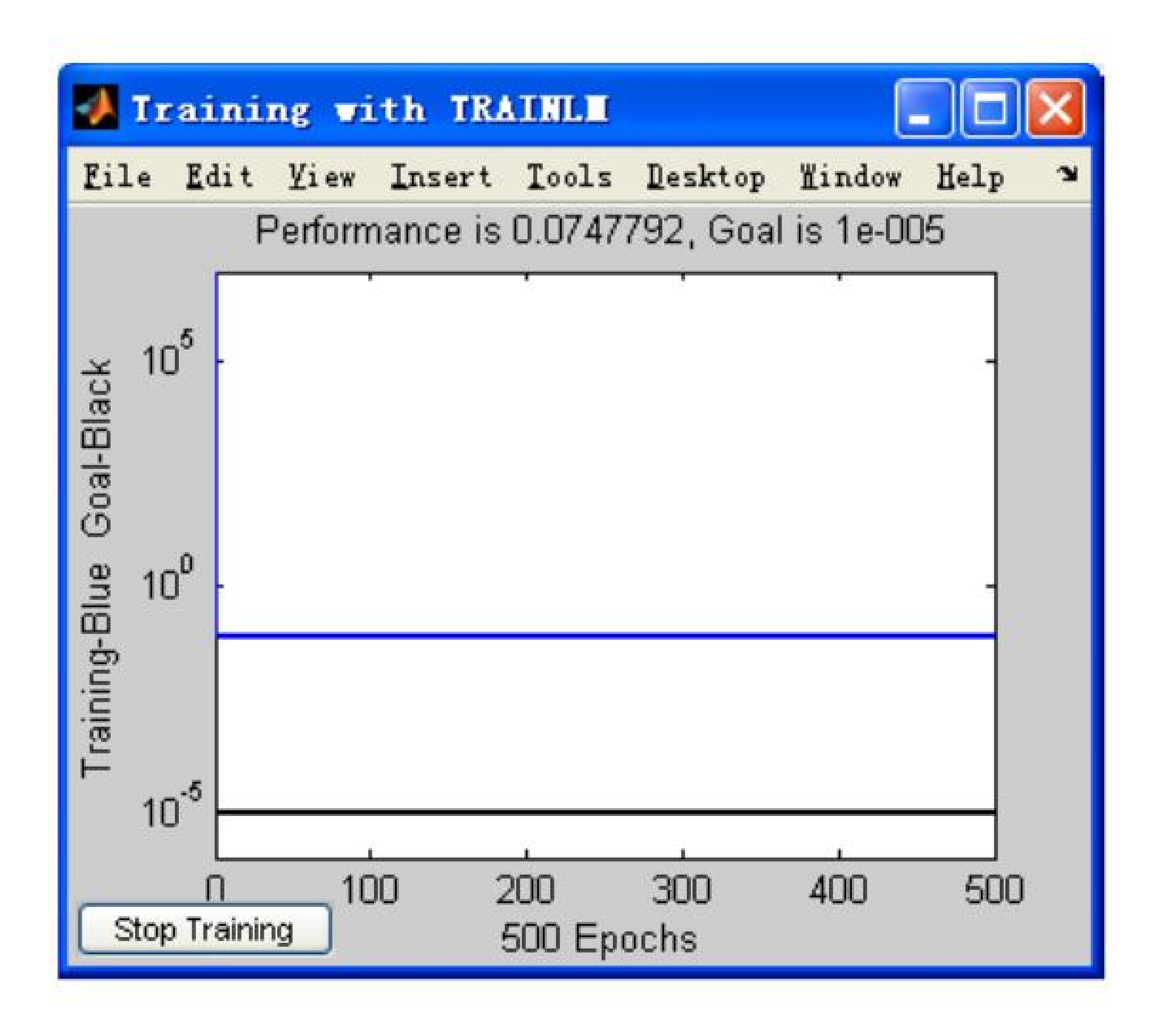


图 4-7 传统 BP 神经网络训练过程误差

4.5.4 小波神经网络在 1-D 插值上的应用

1．案例描述

小波神经网络应用于 1-D 插值。

2．代码实现

```
function retstr = wnndemo()
NNTWARN OFF
retstr=-1;
%
% 加载训练数据
x = (-1:.05:1)';
%
% 输出
y = (-2.186 * x - 1.2864) .* (x < -0.2) + 4.246 * x.* ((x >= -0.2) & (x < 0)) ...
    + (exp(-0.5 .* x - 0.5) .* sin((3 .* x + 7) .* x)) .* (x > 0);
```

```
%
% 绘制输入输出关系图
clf; plot(x,y,'o');
xlabel('x'); ylabel('y'); title('输入输出关系');
%
% 插值
% 可以在网络中设置小波数或者用 FPE 规则在线选择
% 训练过程
Key = keymenu('Make your choice','Use 9 wavelets','On-line choice');
if Key==1
   th = wnetreg(y, x, 9, 4, 3, 1, 5);
   % Use 9 wavelets, 4 iterations, initialization mode 3,
   % minimum number of covered data points 1, scanned levels 5.
else
   th = wnetreg(y, x, [], 4, 3, 1, 5);
end
%
% 对结果进行检测和评价
xg = (-1:.01:1)';
yg = wavenet(xg, th);
%
% 绘制插值结果
plot(x,y,'o', xg,yg,':'); xlabel('x'); ylabel('y'); title('插值结果');
```

3．程序运行

将小波神经网络工具箱 wnet 拷贝到 MATLAB 的 work 目录下，打开 wnndemo 文件，直接运行，主要界面如图 4-8~图 4-12 所示。

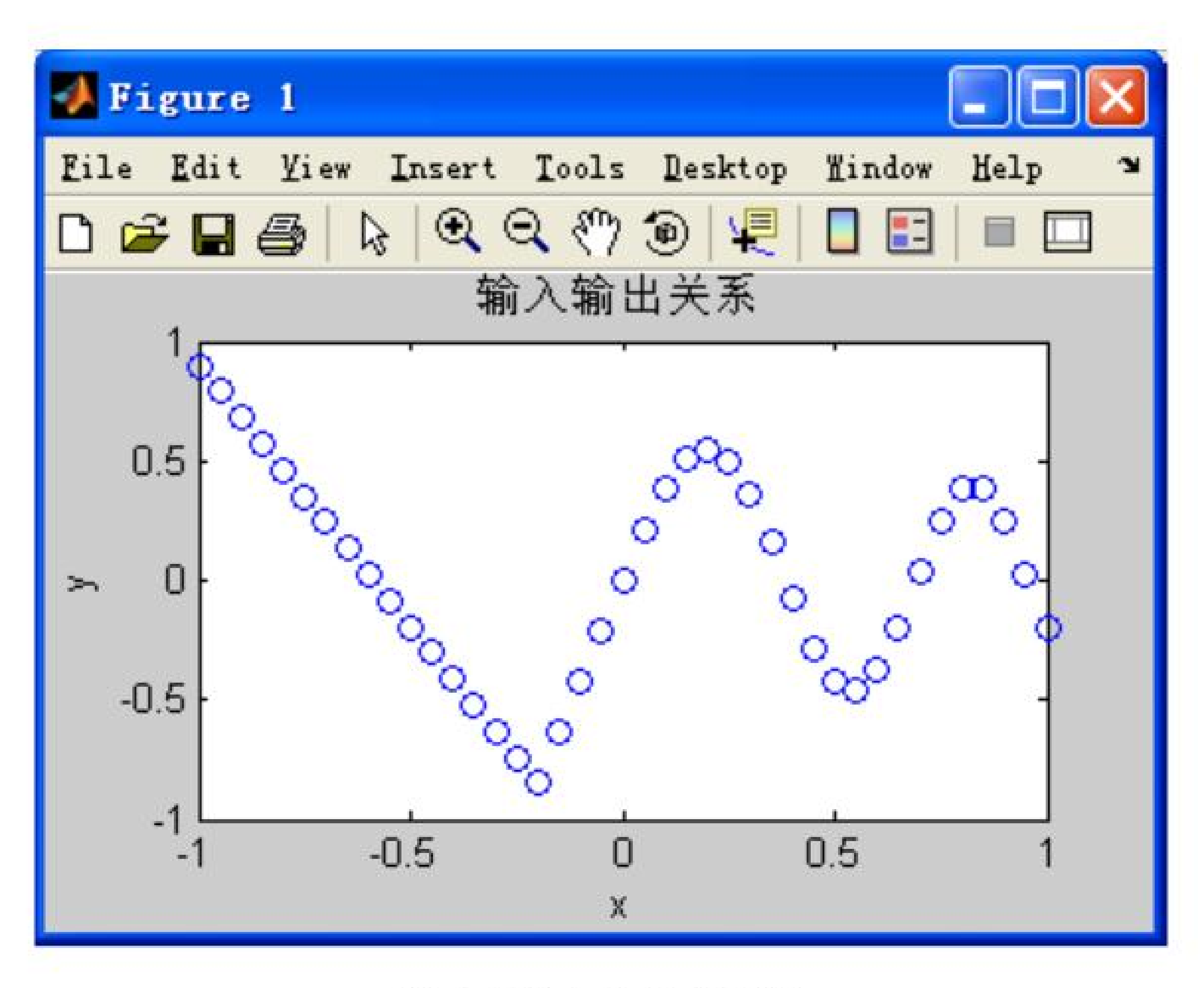


图 4-8 输入输出关系图

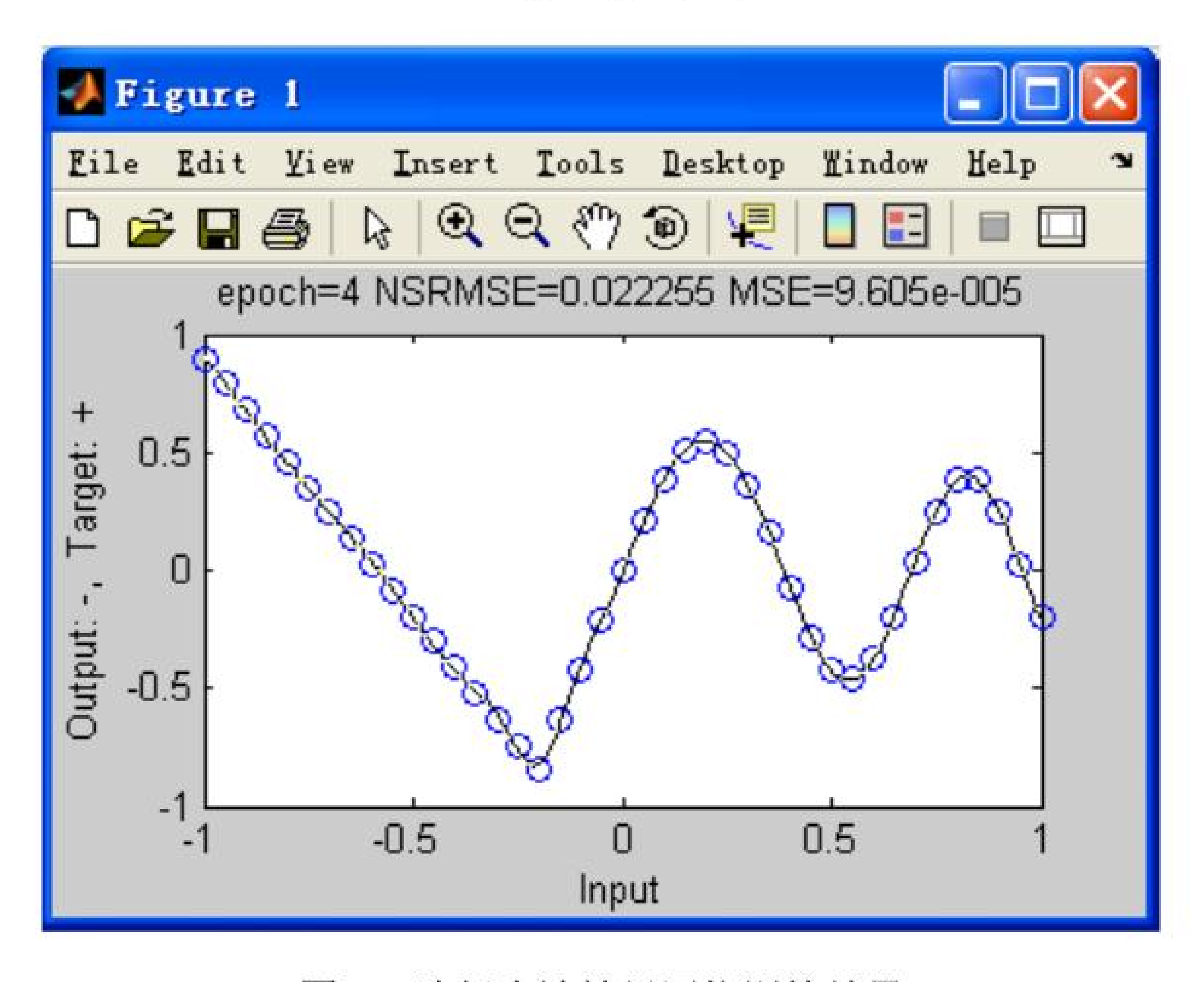


图 4-9 选择小波神经网络训练结果

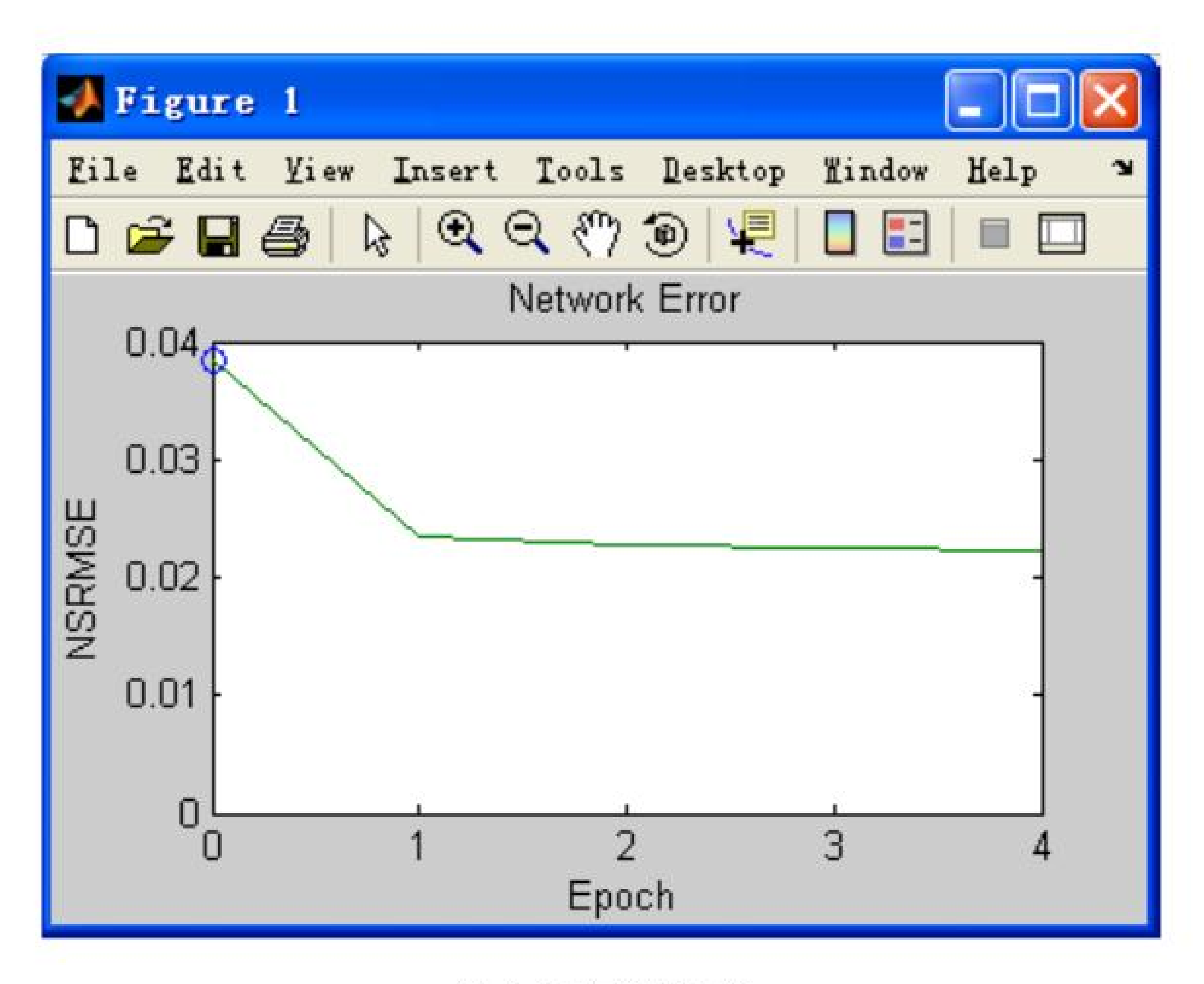


图 4-10 网络误差

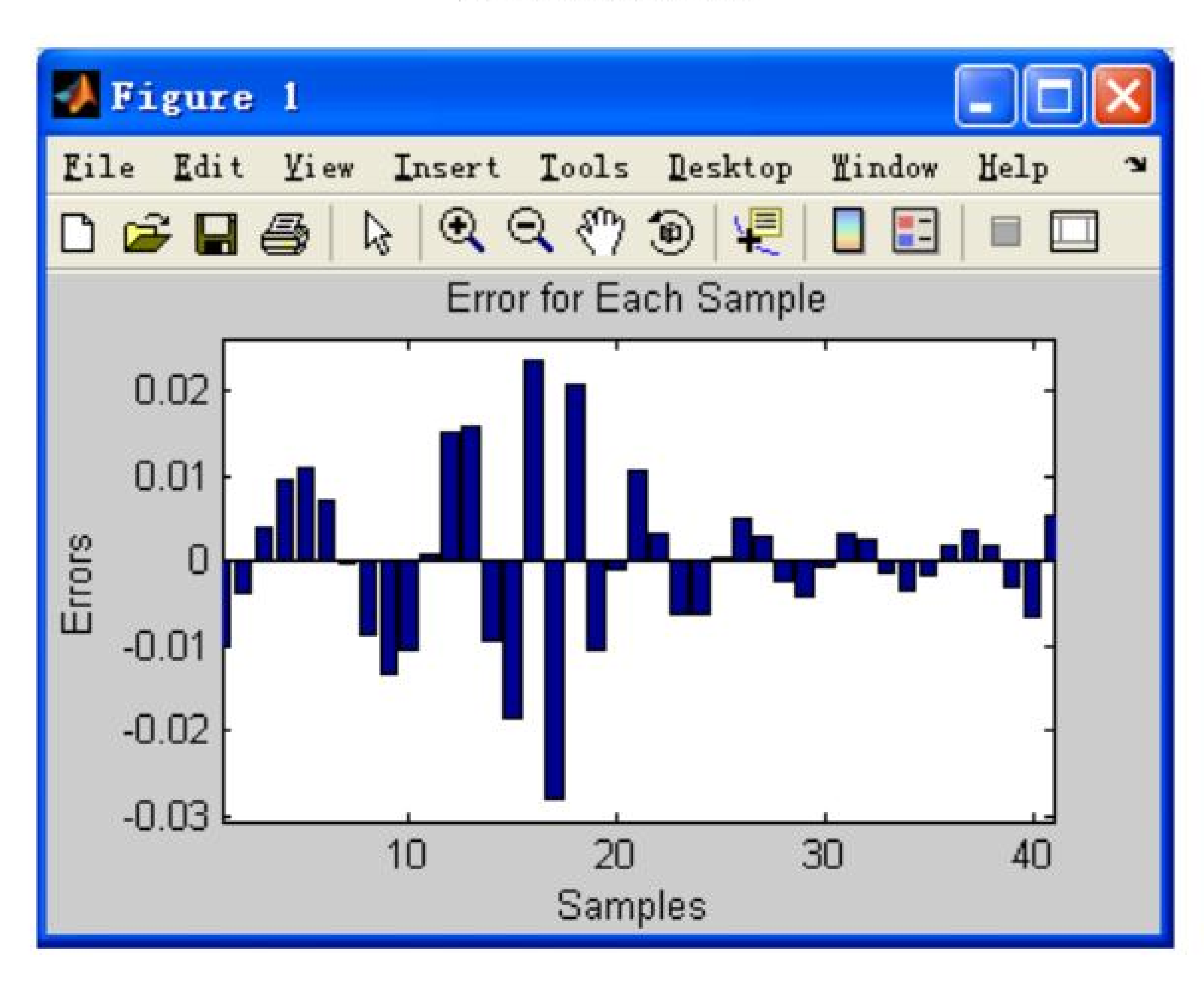


图 4-11 单个样本误差

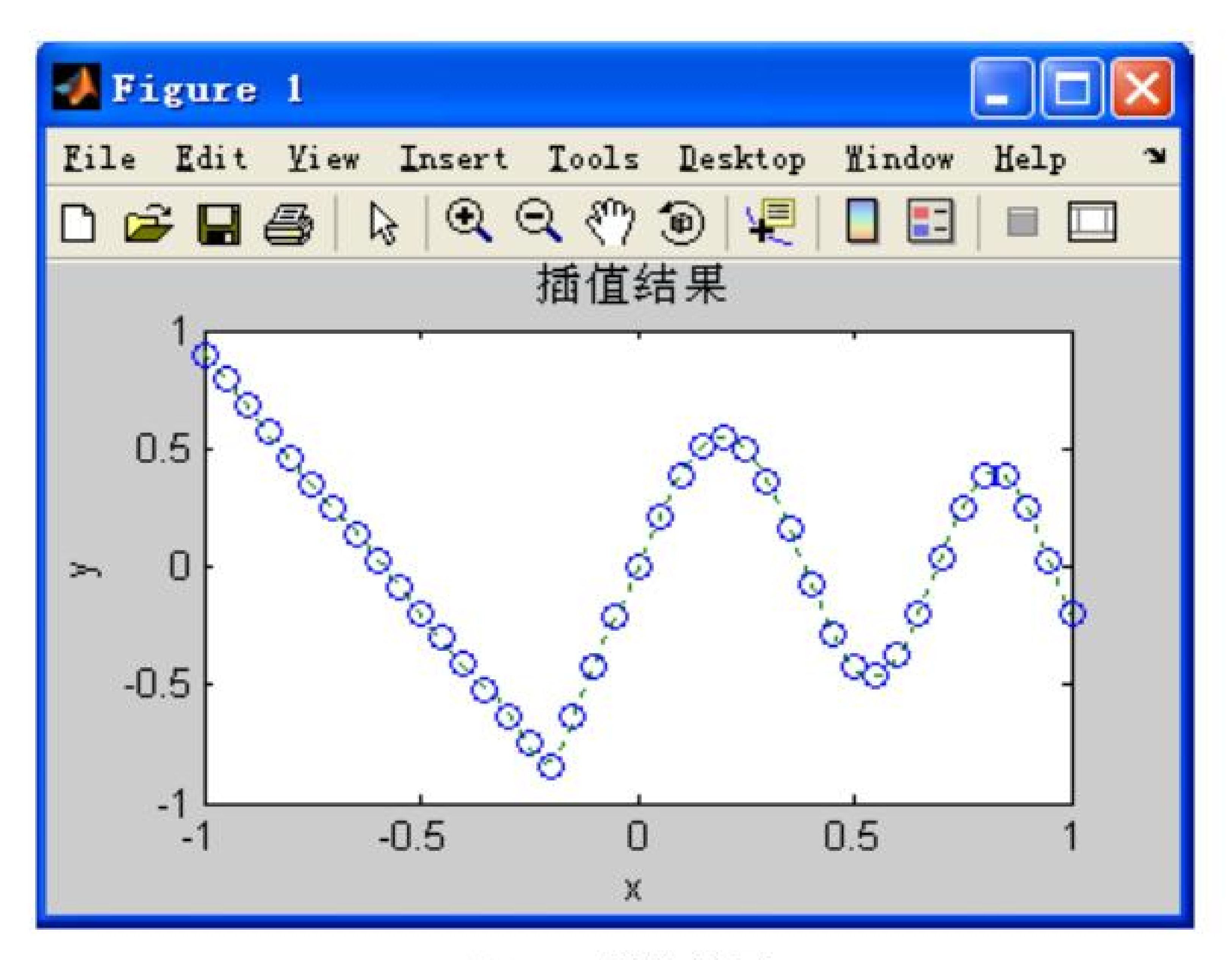


图 4-12 插值结果

练习题

1、谈谈你对 nnToolKit 神经网络工具包的认识。

2、通过什么方法可以在其它高级语言中调用 nnToolKit 工具包函数？

3、请分别用遗传算法优化神经网络权值算法、小波神经网络算法实现 4.5.1 节所示的房地产开发风险预测，并对其结果进行比较。

4、结合某一实际案例，对 nnToolKit 神经网络工具包进行扩展，增加一些自定义函数，从而实现对案例的训练与仿真。

第 5 章 MATLAB 混合编程技术

5.1 概述

MATLAB 是一款强大的工程计算和仿真软件，其中的神经网络工具箱提供了大量可直接调用的函数和命令，基本上囊括了目前应用比较成熟的神经网络设计方法。用 MATLAB 来编写各种网络仿真和训练程序，可以使用户从繁琐的编程中解脱出来，大大提高工作效率。

基于 MATLAB 的工具箱函数实现神经网络混合编程的方法很多，如：基于 Matlab C/C++ 数学库、基于 Matlab Engine、基于 ExcelLink、基于 COM/Excel 生成器、基于 Matlab web server、基于自动化链接、基于 Matlab 的 DDE 功能，都可以实现混合编程。前一章介绍了 nnToolKit 神经网络工具包，它就是在 MATLAB 神经网络工具箱基础上开发的一组神经网络算法函数库，它能方便打包成 DLL 组件，供支持 COM 的高级语言所引用。正因为利用 MATLAB COM 生成器和 Excel 生成器，可以快速实现基于 MATLAB 的混合编程，实现复杂的算法应用。本章将主要介绍如何利用 MATLAB 的 COM Builder 和 Excel Builder 来创建我们自已的组件，以及如何在高级语言中调用这些组件。

5.2 COM 生成器（COM Builder）

COM（Component Object Model，组件对象模型）是以组件为发布单元的对象模型，是一系列面向对象技术和工具的集合。由于 COM 是建立在二进制级别上的规范，所以组件对象之间的交互规范不依赖于任何特定的开发语言。使用该集合，软件开发人员可以用不同厂商提供的组件集成他们自己的应用程序。

从 6.5 版开始，MATLAB 提供了 COM 生成器。COM 生成器提供了实现 MATLAB 独立应用的一种新途径。它能把 MATLAB 开发的算法做成组件，这些组件作为独立的 COM 对象，可以直接被 C++Builder、Visual Basic 或其他支持 COM 的语言所引用。下面详细介绍如何利用 COM Builder 生成器实现对上一章介绍的 nnToolKit 工具包进行封装。

5.2.1 创建 nnToolKit 的 COM 组件

用 MATLAB COM 生成器创建 COM 组件是一个简单的过程，只需要 4 个步骤，即创建工程、管理 M 文件和 MEX 文件、生成组件、打包和分发组件。

1．创建 nnToolKit 工程

在 MATLAB 命令行中输入命令 comtool，打开“MATLAB COM Builder”对话框，它是 MATLAB COM 生成器的主要工作环境。

在“File”菜单中选择“New Project”选项，将弹出“New Project Setting”对话框，如图 5-1 所示。

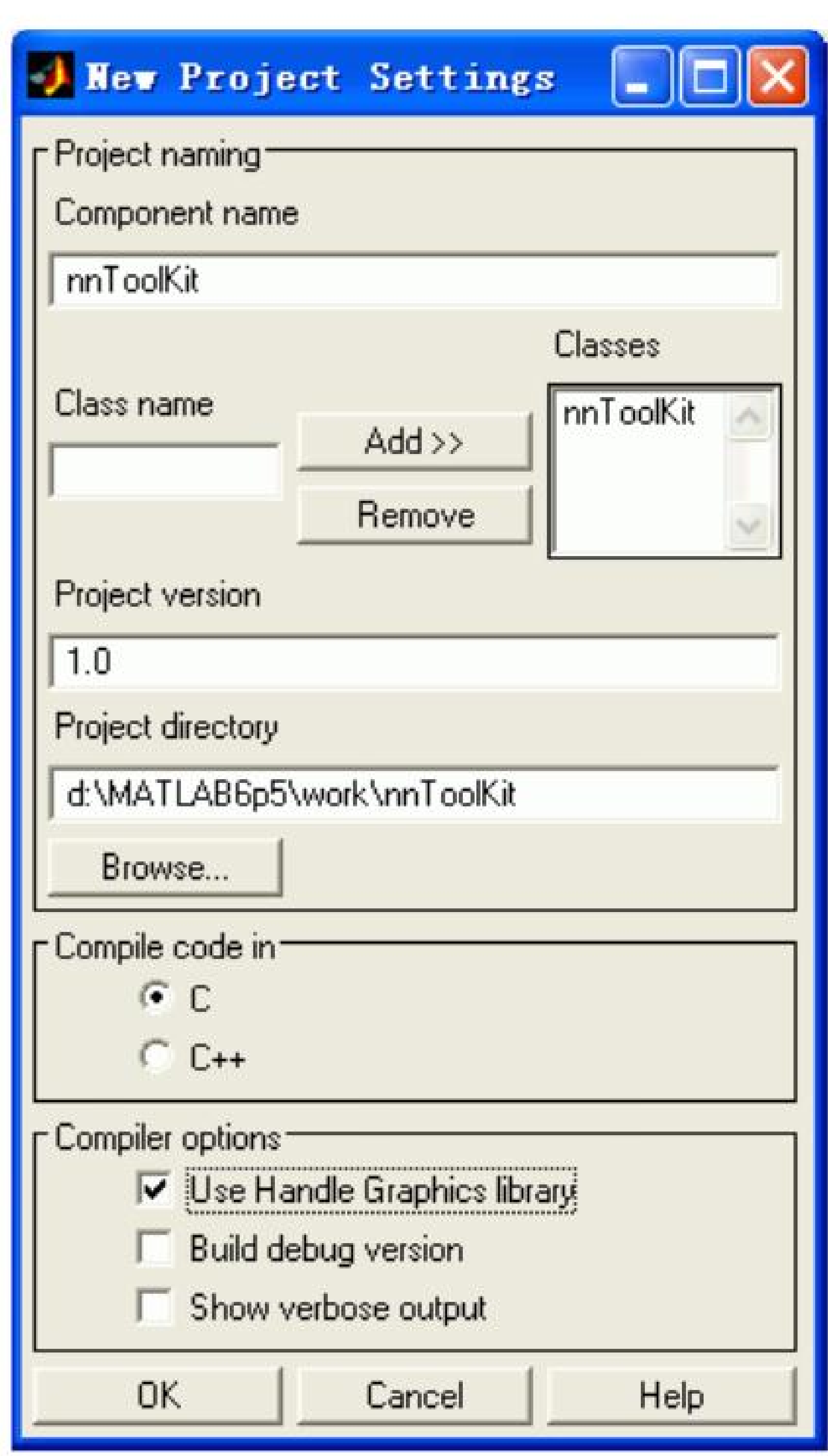


图 5-1 创建 nnToolKit 工程

在“Component name”文本框中输入组件名 nnToolKit（DLL 文件），该组件名输入后，生成器自动在“Class name”文本框中产生一个同名的类名 nnToolKit。“Project version”文本框中为版本号，默认为 1.0。“Project directory”文本框为在编译和打包模型时，工程和相关文件的存放位置，这里为“D:\MATLAB6p5\work\nnToolKit”。由于神经网络训练时要用到 MATLAB 的图形库，故在工程设置界面中要选中“Use Handle Graphics library”复选框，其它默认设置即可。点击“OK”按钮，将在指定目录下自动创建一个新的工程文件 nnToolKit.cbl，以后可以通过“File/Open Project”操作打开已生成的 nnToolKit.cbl 项目文件，来修改该工程。

2．管理 M 文件（神经网络相关函数）

单击“Add File”按钮或从“Project”菜单中选择“Add File…”选项，将已调试好的神经网络相关函数（LmTrain.m、LmSimu.m 等）加入到项目中。

3．生成 nnToolKit 组件

定义好工程设置并添加了相关神经网络函数后，通过“Build”菜单中的“COM Object”选项或直接单击“Build”按钮来调用 MATLAB 编译器，把中间源文件写到 D:\MATLAB6p5\work\nnToolKit\src 目录中，将必要的输出文件写到 D:\MATLAB6p5\work\nnToolKit\distrib 目录中，“Build Status”面板显示生成过程的输出，如图 5-2 所示。

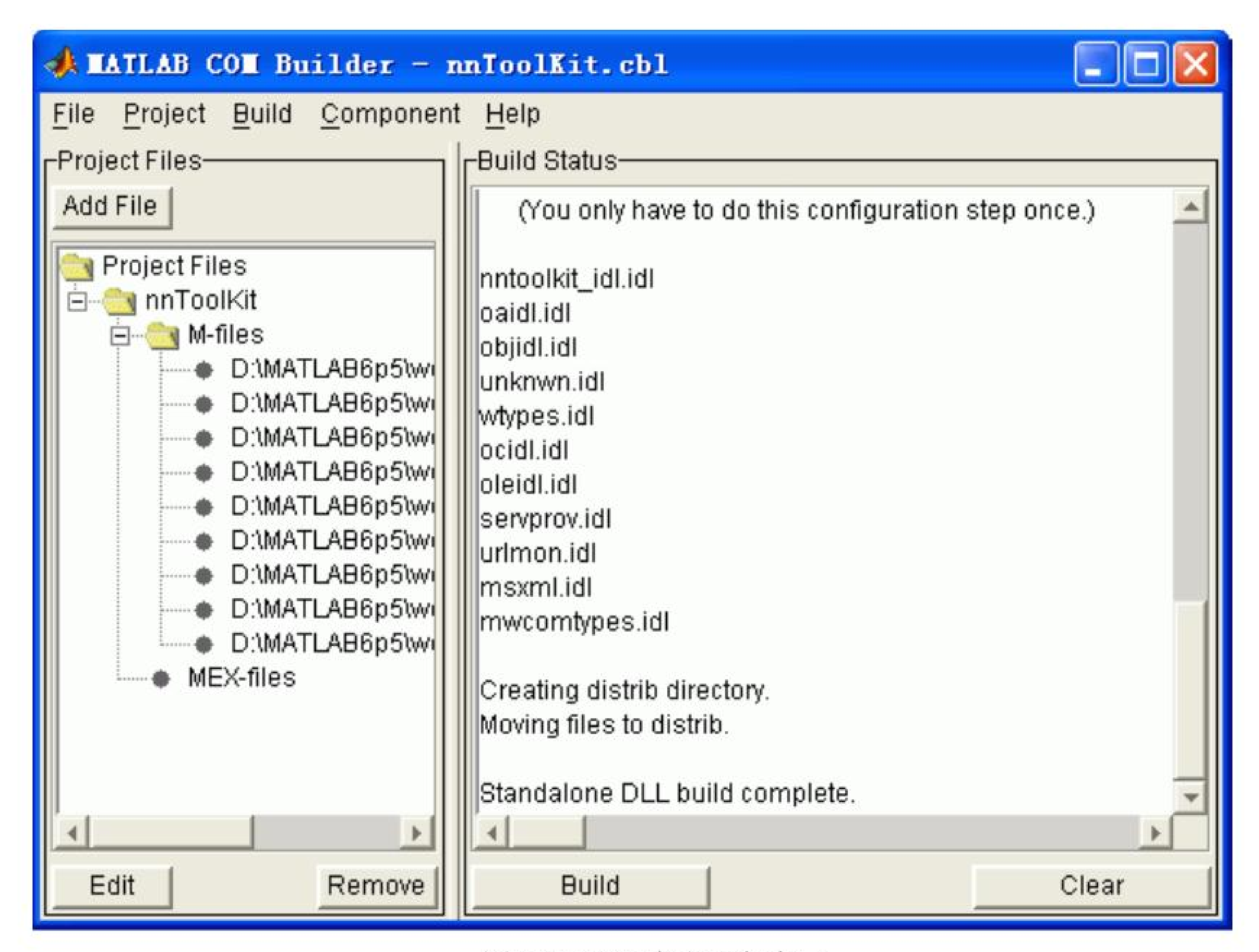


图 5-2 COM 生成器主窗口

4．打包和分发组件

一旦模型编译成功并进行了测试，就可以打包并分发了。从“Component”菜单中选择“Package Component”选项，将创建包含如表 5-1 所示文件的自解压可执行程序。

表 5-1 自解压文件 nnToolKit.exe 包含的文件

文 件	功 能
_install.bat	由自解压可执行程序运行的脚本
nnToolKit_1_0.dll	编译后的组件
mglinstaller.exe	MATLAB 数学库和图形库安装器

Mwcomutil.dll	COM 生成器工具库
mwregsvr.exe	在目标机器上注册 DLL 可执行程序

自解压可执行程序的名称为 nnToolKit.exe。在计算机上运行安装器，将按以下步骤进行：

1) mglinstaller 安装 MATLAB C/C++数学库和图形库。
2) 添加 mglinstaller 创建的目录<application>\bin\win32 目录到环境变量 path 中，其中<application>表示配置应用的根目录。
3) mwregsvr 注册 mwcomutil.dll 和 nnToolKit_1_0.dll。

必须在每个需要安装组件的计算机上重复本发布过程，下面详细介绍组件的安装和使用。

5.2.2 nnToolKit 组件的安装

直接运行 nnToolKit.exe 自解压安装文件，进入图 5-3 所示的界面：

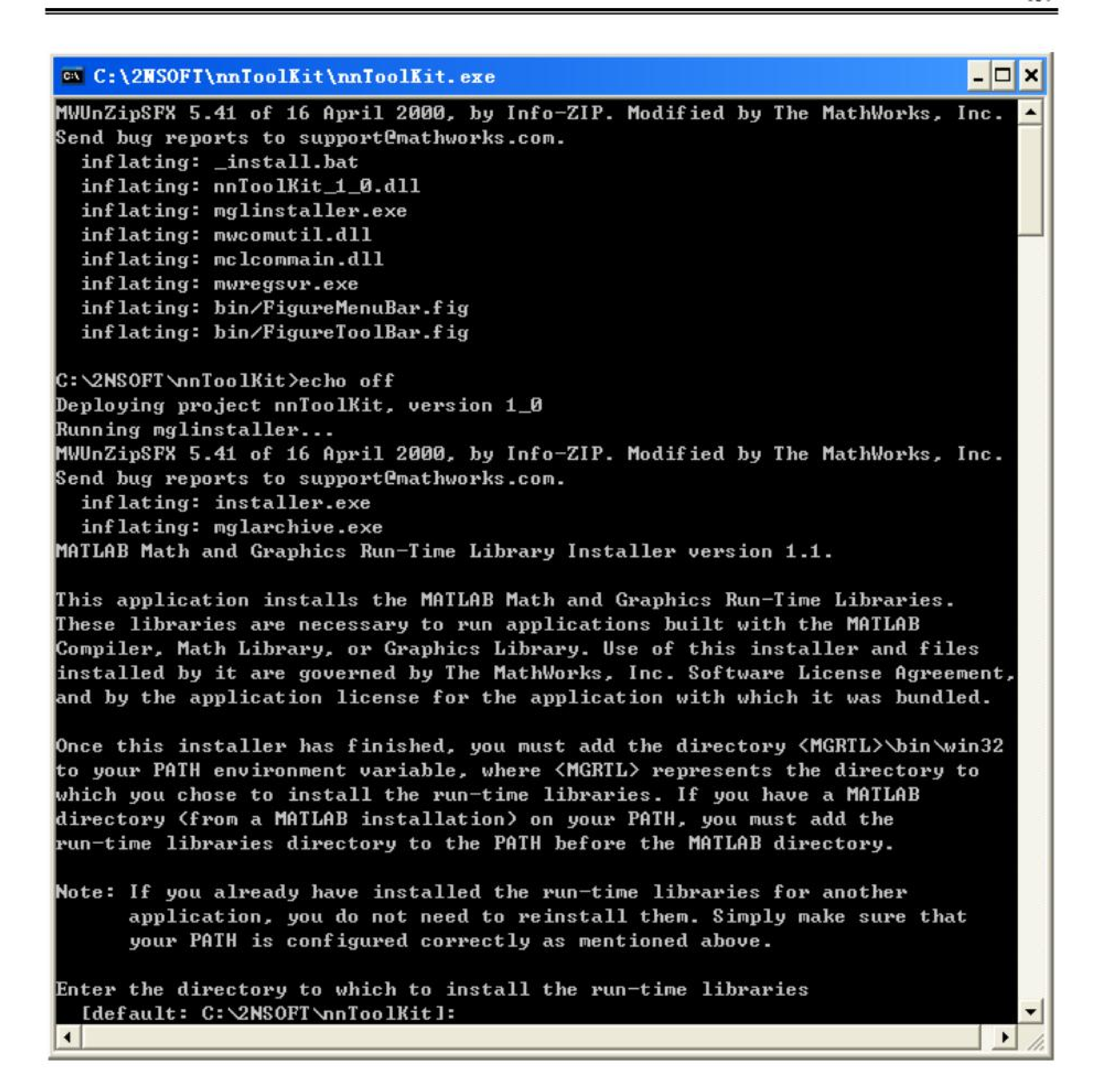


图 5-3 nnToolKit 安装

这里安装文件首先完成自解压，然后提示组件安装路径，用户可以直接回车，即默认安装，将

软件安装在当前安装文件所在的路径，也可以另外指定一个安装路径。这里，选择安装到：C:\2NSOFT\nnToolKit。按回车继续，进入如图 5-4 所示的界面。

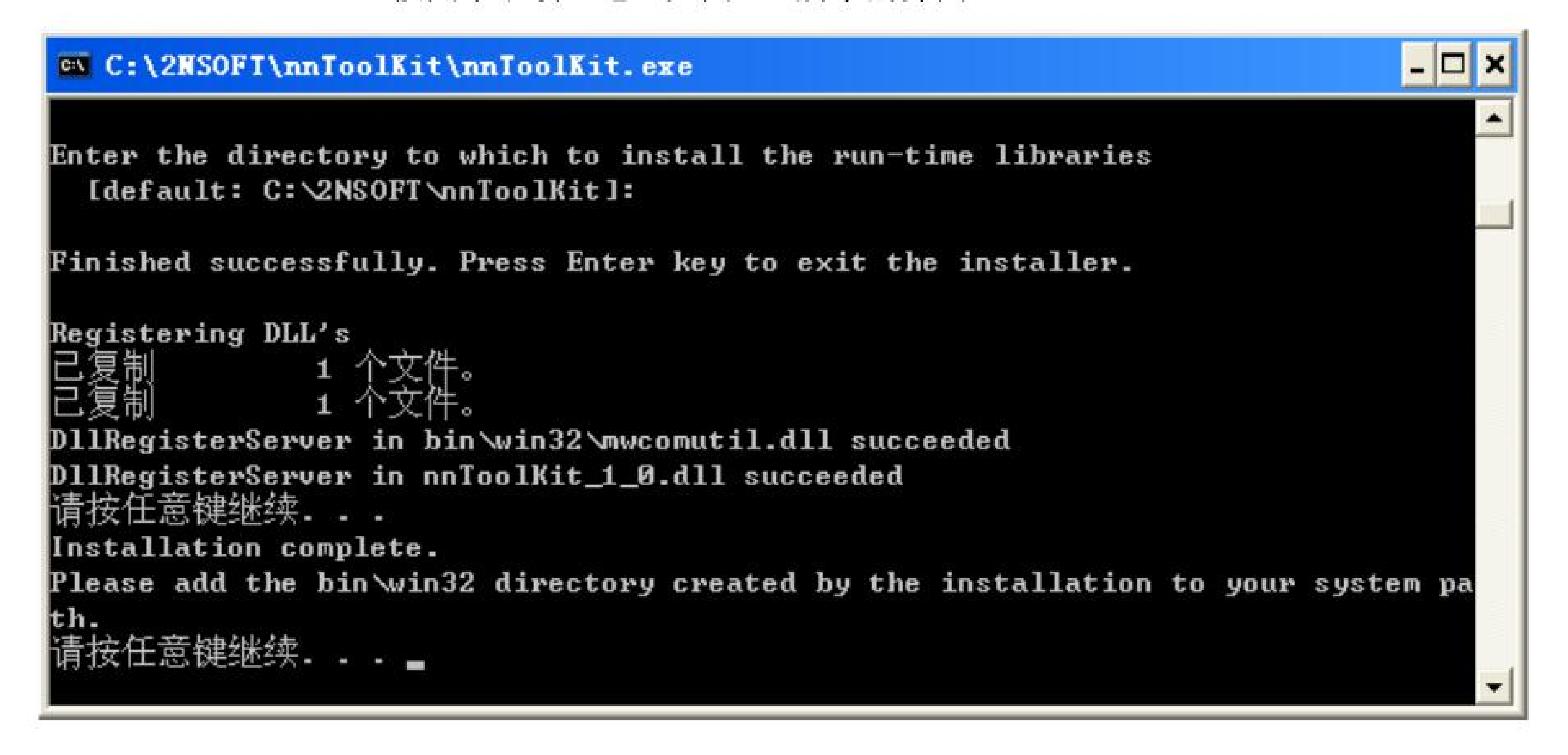


图 5-4 提示 COM 组件注册和设置环境变量

按提示继续操作，完成相关组件的注册，直至 nnToolKit 工具包安装成功。根据系统安装时的提示，还需要手工配置环境变量，具体操作为：

进入我的电脑→属性→环境变量，将“C:\2NSOFT\nnToolKit\bin\win32”加至 path 路径中。如图 5-5 所示。



图 5-5 设置环境变量

5.2.3 VB 调用 nnToolKit 神经网络工具包实现混合编程

在 VB 环境下调用 nnToolKit 神经网络工具包，开发神经网络应用系统，主要分为以下几个步骤：

1．创建神经网络应用工程。

打开 Visual Basic 开发环境，并创建一个新的工程，取名为 NetProj。在 NetProj 工程中，创建两个模板文件 CBpnFile.bas，CNetPara.bas 和三个窗体表单文件 frmAbout.frm，frmLmNet.frm，frmUnitary.frm，其中 frmLmNet.frm 为系统主界面，给空白的窗体中添加控件，如图 5-6 所示，并按

表 5-2 中的内容设置控件属性。

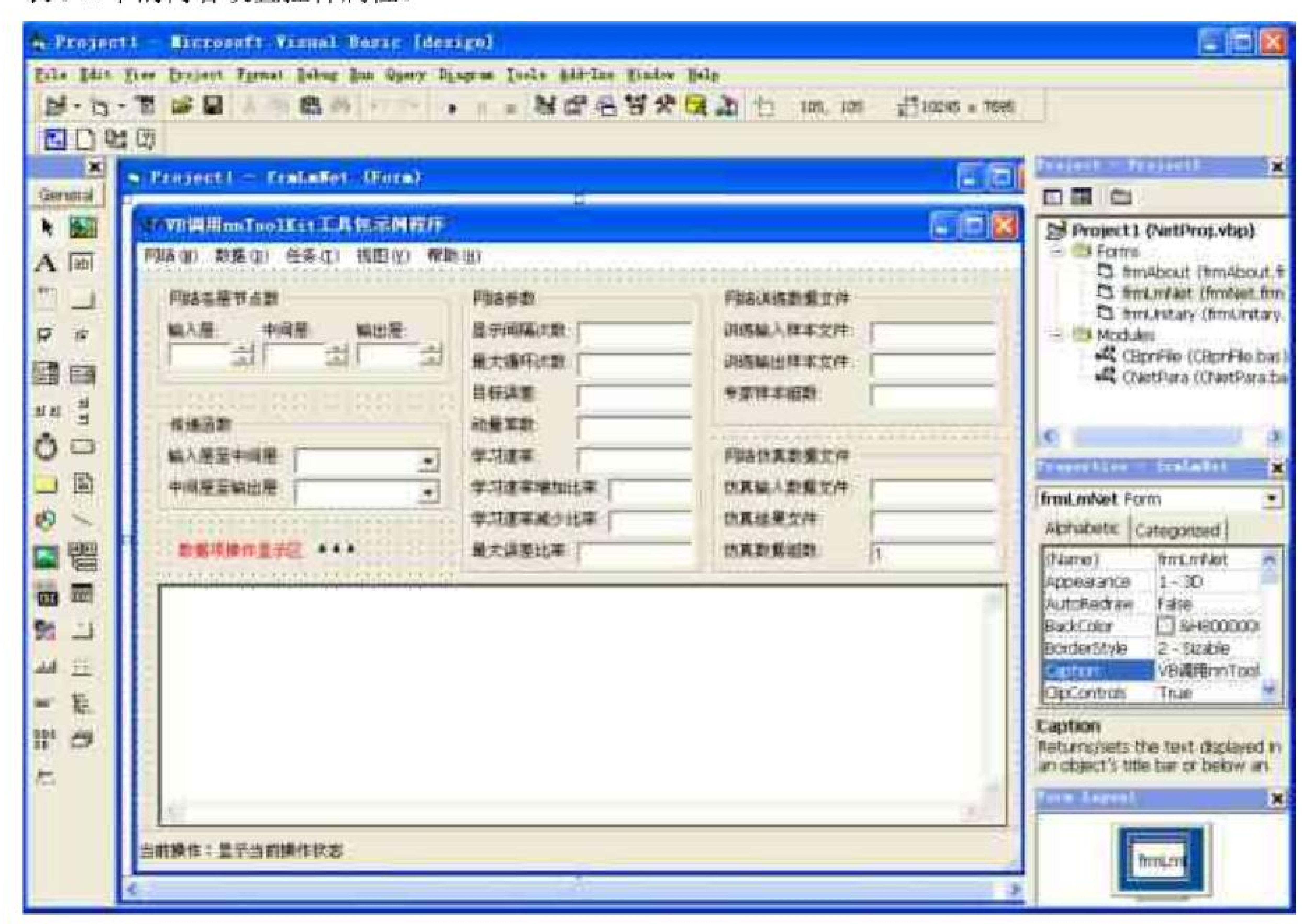


图 5-6 创建 frmLmNet 窗体

表 5-2 主要属性设置

控件类型	控件名称	属　性
Frame	fNodeNum	神经网络各层节点数设置框
TextBox	txtNode(0)	输入层节点数
TextBox	txtNode(2)	中间层节点数

TextBox	txtNode(1)	输出层节点数
Frame	fNetFunc	神经网络传递函数设置框
ComboBox	combFunc(0)	输入层至中间层传递函数
ComboBox	combFunc(1)	中间层至输出层传递函数
Frame	fNetPara	神经网络训练参数设置框
TextBox	txtTrainPara(0)	显示间隔次数
TextBox	txtTrainPara(1)	最大循环次数
TextBox	txtTrainPara(2)	目标误差
TextBox	txtTrainPara(3)	动量常数
TextBox	txtTrainPara(4)	学习速率
TextBox	txtTrainPara(5)	学习速率增加比率
TextBox	txtTrainPara(6)	学习速率减少比率
TextBox	txtTrainPara(6)	最大误差比率
Frame	fTrainDataFiles	神经网络训练数据文件设置框
TextBox	txtTrainIn	训练输入样本文件
TextBox	txtTrainOut	训练输出样本文件
TextBox	txtTrainNum	专家样本组数
Frame	fSimuDataFiles	神经网络仿真数据文件设置框
TextBox	txtSimuIn	仿真输入数据文件
TextBox	txtSimuOut	仿真输出数据文件
TextBox	txtSiumNum	仿真样本组数

2． 在工程中引用 nnToolKit 库。

点击菜单“Project/References”，弹出如图 5-7 所示的对话框：

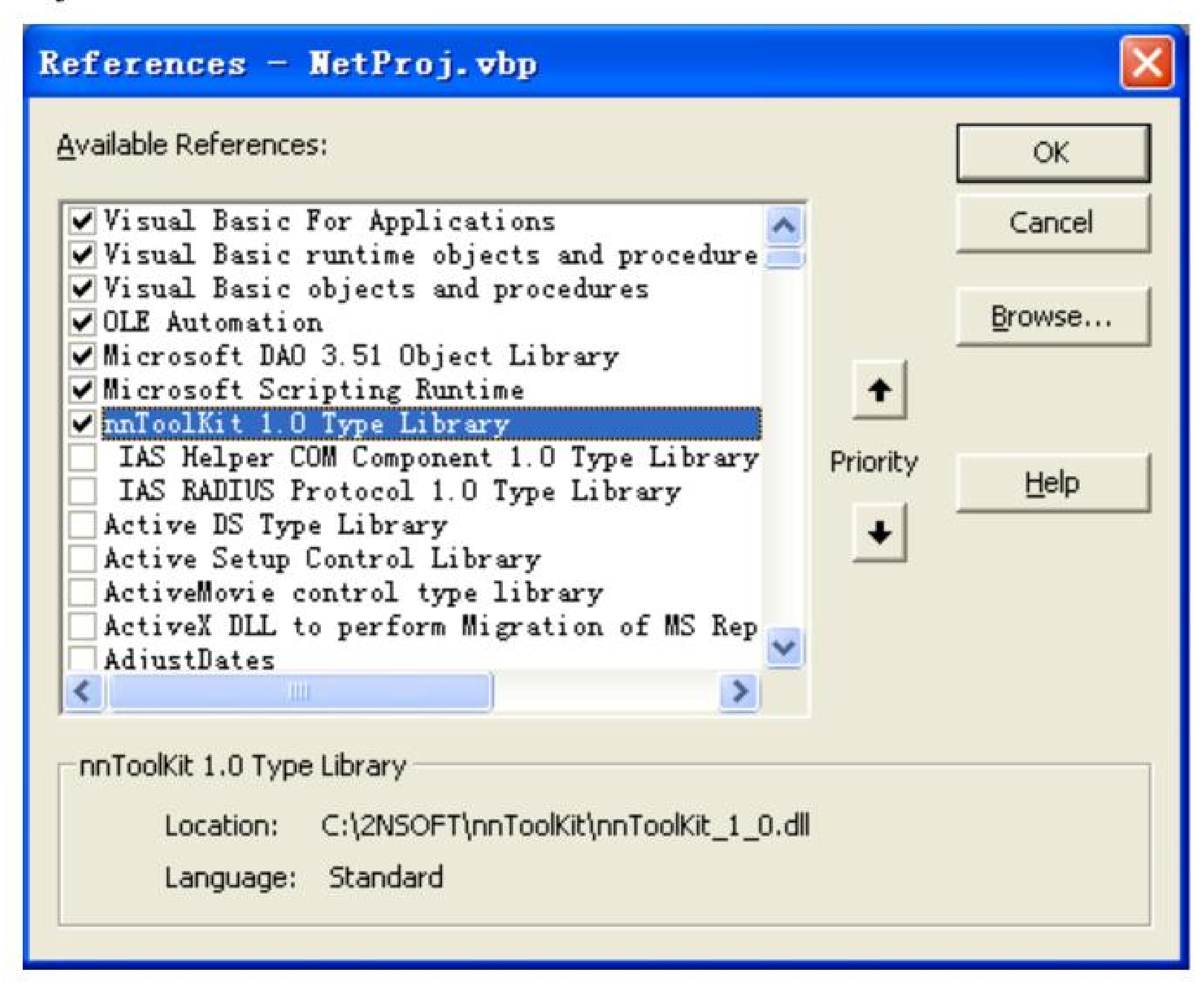


图 5-7 引用 nnToolKit 类型库

如果完成了 nnToolKit 组件的安装（见 5.2.2 节），nnToolKit 1.0 Type Library 将自动加到有效引用列表中，选中 nnToolKit 1.0 Type Library，然后单击 OK 按钮，至此，该库就加入到当前工程中。读者可以尝试在 Visual Basic 环境下，点击“View/Object Browser”，出现如图 5-8 所示的对话框。在<Classes>栏中选中 nnToolKit，在右边的列表框中显示 nnToolKit 类的所有成员函数，选中某个成员函数，从下面的备注框可以发现类里成员函数的参数形式与.M 文件里的参数形式是不同的，这个请读者注意。下面的步骤就是利用 VB 的知识调用这个库。

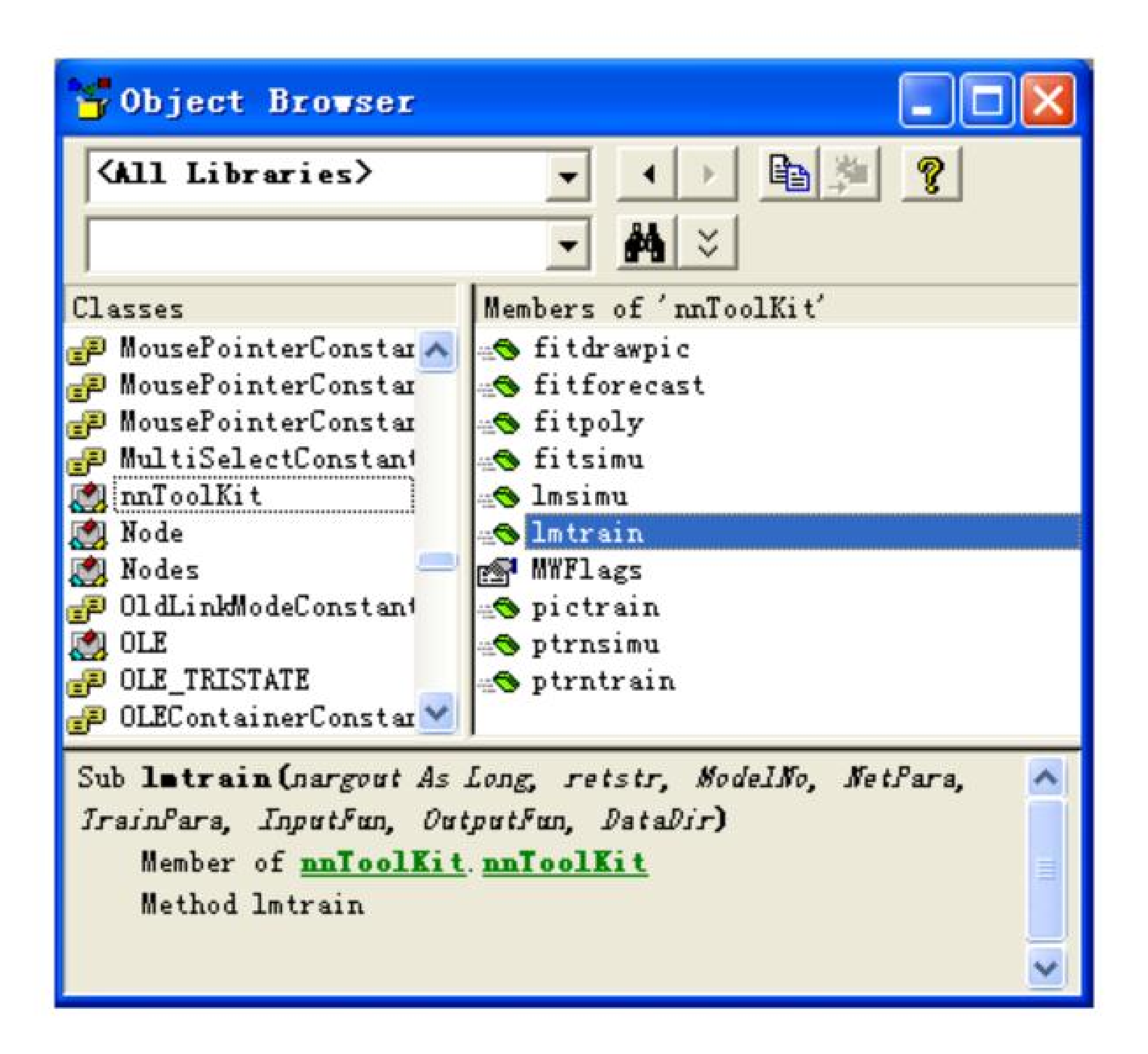


图 5-8 nnToolKit 成员函数

3．代码实现（主要代码）

```
'
'定义 nnToolKit 神经网络工具包的对象实例 LmNet
'
 Private LmNet As nnToolKit.nnToolKit
 Set LmNet = New nnToolKit.nnToolKit
'
'网络训练
'
Private Sub mnuStartTrain_Click()
   On Error GoTo Handle_Error
   '返回值
   Dim retstr As Variant
```

```
   '网络训练基本参数
   Dim NetTPara(4) As Double
   '模型编号
   Dim vModelNo As Variant
   vModelNo = ModelNo
   For i = 0 To 2
      NetTPara(i) = Nodes(i)
   Next
   NetTPara(3) = TrainSampleNum    '样本个数
   '
   '网络训练调用参数说明
   '1，一个返回参数
   'retstr，返回值
   '网络模型基本参数，Variant
   '神经网络参数，Double，包括输入层节点数；输出层节点数；中间层节点数；网络训练
样本个数。
   '神经网络训练参数，若为“-1”表示选择默认参数
   '网络仿真时输入层至中间层的传递函数
   '网络仿真时中间层至输出层的传递函数
   '程序运行时的当前目录
   Call LmNet.lmtrain(1,retstr,ModelNo,NetTPara,-1,TransFunc(0),TransFunc(1),FilePath)
   MsgBox ("训练完成，您现在可以进行仿真操作了！")
   OperStatus_Show ("训练完成，您现在可以进行仿真操作了！")
   Exit Sub
Handle_Error:
   MsgBox (Err.Description)
End Sub
'
'网络仿真
'
Private Sub mnuTest_Click()
   '神经网络仿真参数
   Dim SimulatePara(7) As Double
   '模型编号
   Dim ModelNo As String
```

```
  ModelNo = "1"
  Dim fso As New FileSystemObject，  fil As File，  ts As TextStream
  On Error GoTo Handle_Error
  Set fil = fso.GetFile(FilePath + InSimuFile)
  Set ts = fil.OpenAsTextStream(ForReading)
  DataLine = ts.ReadLine
  DataLength = Len(DataLine)
  For i = 0 To Nodes(0) - 1
    SimulatePara(i) = Trim(Left(DataLine，  5))
    DataLength = Len(DataLine)
    DataLine = Trim(Right(DataLine，  DataLength - 5))
  Next
  ts.Close
  '
  '网络仿真调用参数说明
  '1，一个返回参数
  'retstr，返回值
  '网络模型基本参数，Variant
  '神经网络参数，Double，包括输入层节点数；输出层节点数；中间层节点数
  '神经网络仿真参数，若为"-1"表示选择默认参数
  '网络仿真时输入层至中间层的传递函数
  '网络仿真时中间层至输出层的传递函数
  '程序运行时的当前目录
  Call LmNet.lmsimu(1,retstr,ModelNo,Nodes,SimulatePara,TransFunc(0),TransFunc(1),
FilePath)
  OperStatus_Show ("仿真成功，点击'数据/显示测试结果'查看仿真结果！")
  MsgBox ("仿真成功，点击'数据/显示测试结果'查看仿真结果！")
  Exit Sub
Handle_Error:
  MsgBox (Err.Description)
End Sub
```

5.2.4 CB 调用 nnToolKit 神经网络工具包实现混合编程

在 CB 环境下调用 nnToolKit 神经网络工具包，开发神经网络应用系统，主要分为以下几个步骤：

1．在 CB 中创建基于 nnToolKit 的自定义 VCL 组件

关于 CB 中如何创建自定义 VCL 组件，请读者参看 CB 相关的资料介绍，在此不再另述，在图书配套的源程序中，已经建有一个 VCL 工程，打开项目文件 nnToolKit.bpk，出现图 5-9 所示的对话框：

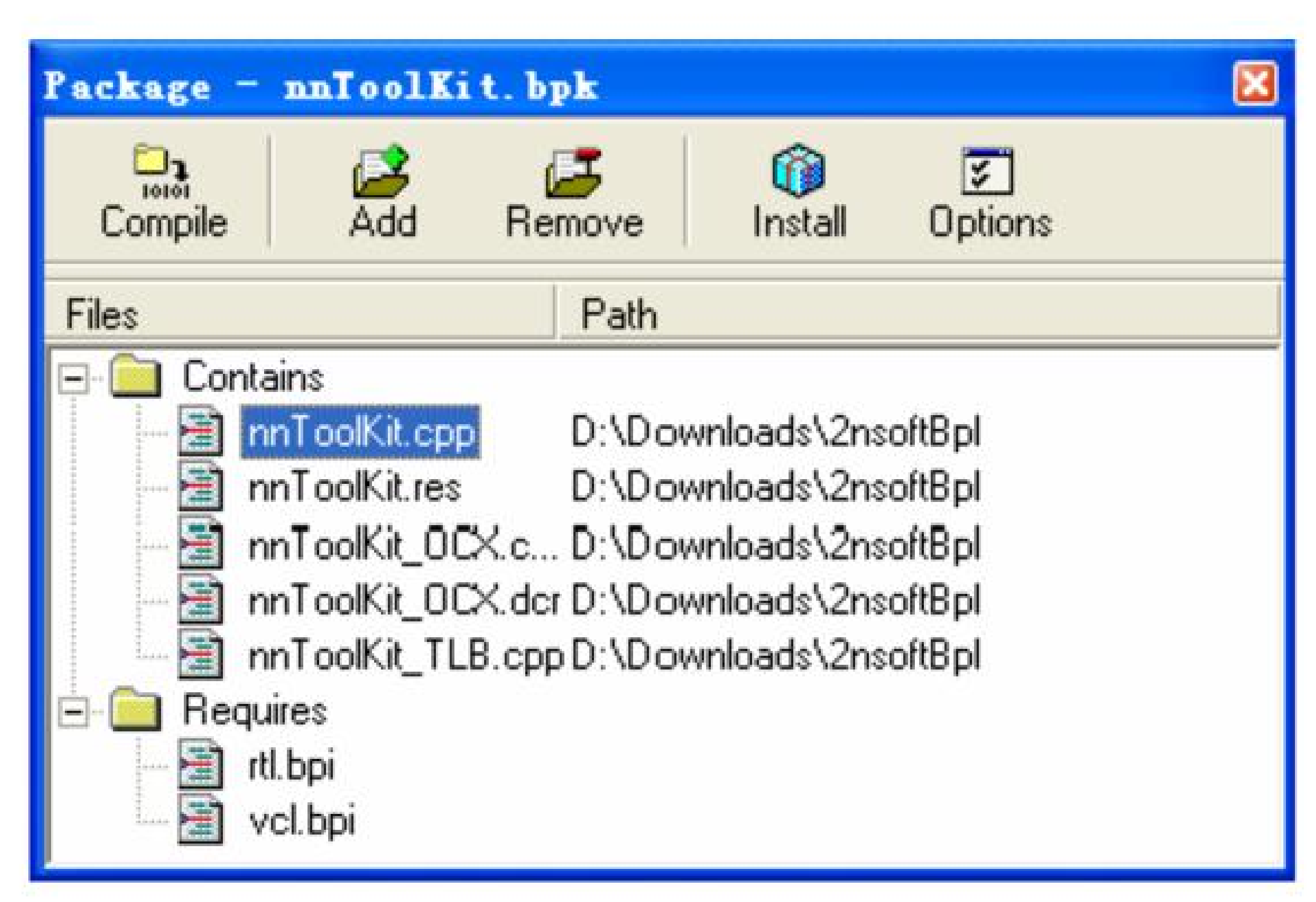


图 5-9 创建自定义 VCL 组件

编译项目文件后，再点击“install”操作，安装完成后，在当前开发环境下的 ActiveX 的组件页中会出现 nnToolKit ActiveX 控件，如图 5-10 所示：



图 5-10 出现在组件页上的 nnToolKit 控件

2．创建神经网络应用工程

打开 C++ Builder 开发环境，并创建一个新的工程，取名为 NetProj。在主窗体中加载一个 TPageControl 控件，并在其中增加两个页面，分别对应网络训练和网络仿真。分别如图 5-11 和 5-12 所示。

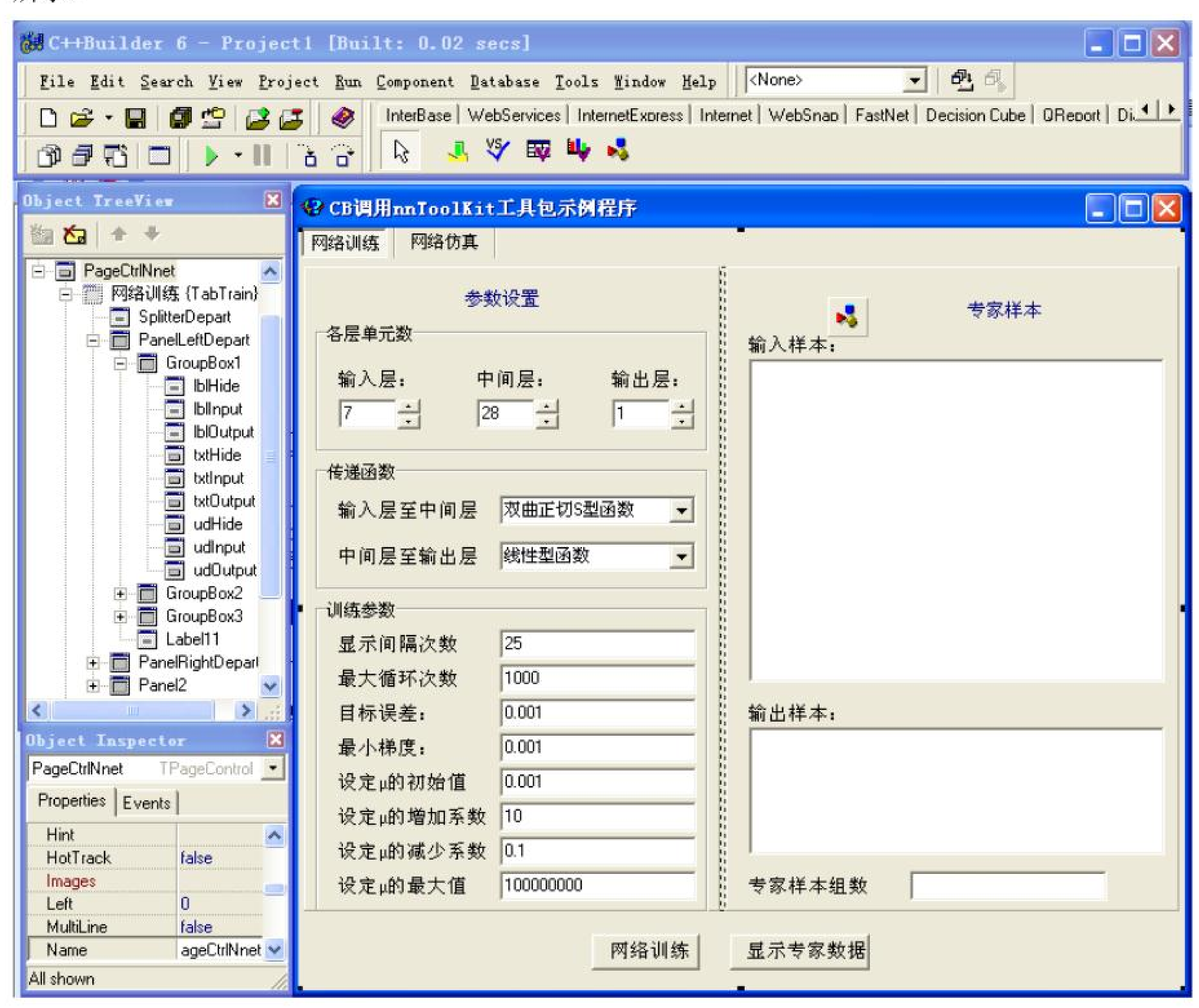


图 5-11 神经网络训练界面

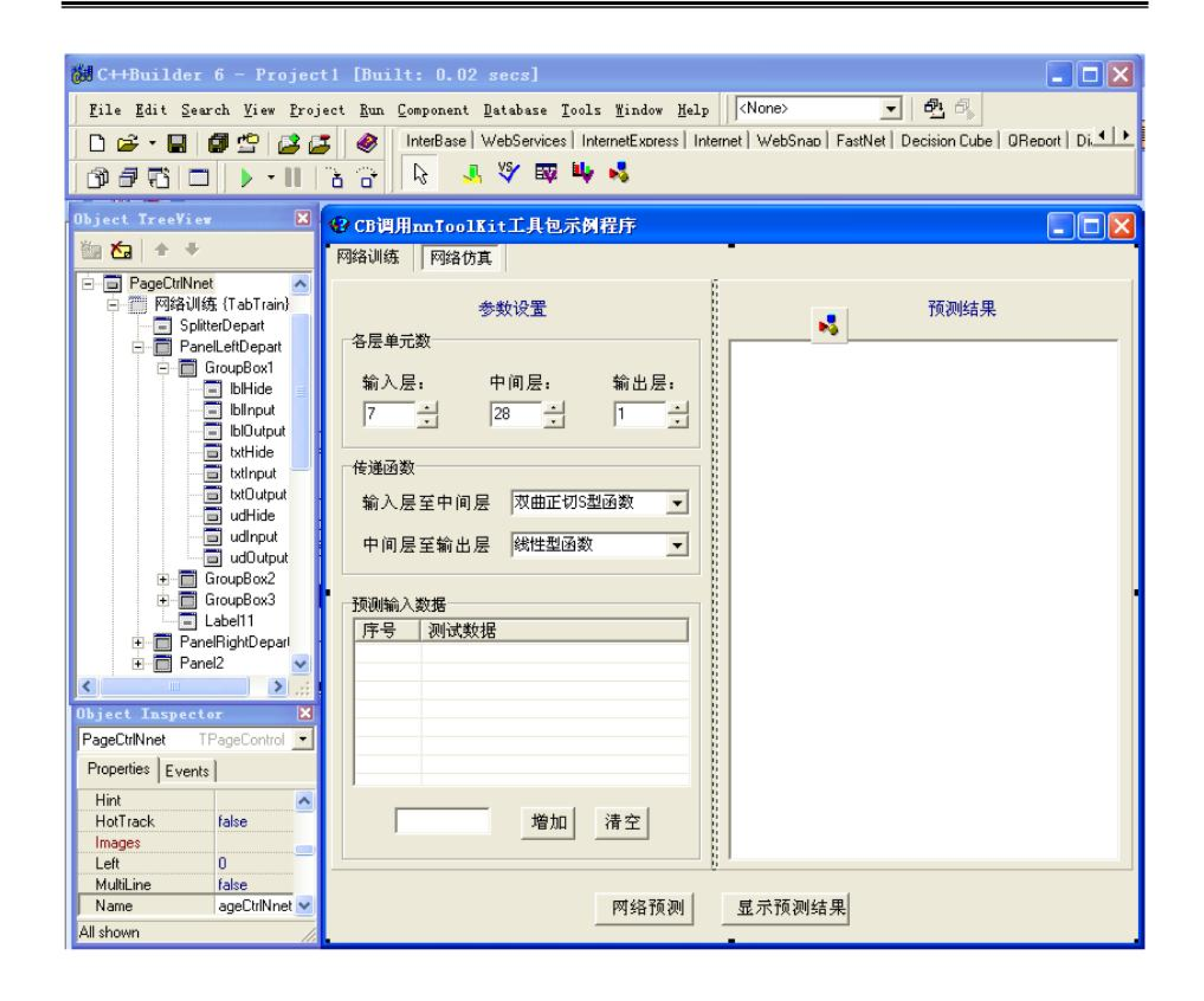


图 5-12 神经网络仿真界面

3．加载 nnToolKit ActiveX 控件

从工具条 ActiveX 的组件页中，将 nnToolKit ActiveX 控件拖至设计窗体中，这时在头文件中将自动加入一行：

#include "nnToolKit_OCX.h"

在构造函数中自动定义一个类 TnnToolKit 对象：

TnnToolKit *nnToolKit1;

在实现文件中也会自动加入一行：

#pragma link "nnToolKit_OCX"

4．代码实现（主要代码）

```
//
//网络训练
//
void __fastcall TfrmNnet::btnTrainClick(TObject *Sender)
{
      try{
            Variant ParaIn1(OPENARRAY(int，(0，4))，varDouble);        //神经网络模型参数
            Variant ParaIn2(OPENARRAY(int，(0，8))，varDouble);        //神经网络训练参数
            TVariant RetValue;                                           //函数返回值
            double NetParaDouble[4]，TrainParaDouble[8]，RetDouble;
            TVariant InputFunc;       //输入层到中间层的传递函数
            TVariant OutputFunc;      //中间层到输出层的传递函数
            TVariant FileDir;         //专家样本数据文件目录
            FileDir = ExtractFilePath(Application->ExeName) +"netdata";
            TVariant ModelNo = "1";           //模型编号
⋮
⋮
nnToolKit1->lmtrain(1,&RetValue,ModelNo,ParaIn1,ParaIn2,InputFunc,OutputFunc,FileDir);
                        ⋮
                        ⋮
      }
catch (Exception &exception)
{
      ShowMessage("神经网络训练出错！");
      return;
}
}
//
//网络仿真
```

```
//
void __fastcall TfrmNnet::btnSimulateClick(TObject *Sender)
{
                ⋮
                ⋮
        Variant ParaIn1(OPENARRAY(int，(0，3))，varDouble)；
        Variant ParaIn2(OPENARRAY(int，(0，iInputNums))，varDouble)；
        TVariant RetValue；
        double NetParaDouble[3]，SimulatePara[7]，RetDouble；
        TVariant InputFunc；      //输入层到中间层的传递函数
        TVariant OutputFunc；      //中间层到输出层的传递函数
        TVariant FileDir；         //专家样本数据文件目录
        FileDir = ExtractFilePath(Application->ExeName) +"netdata"；
        TVariant ModelNo = "1"；       //模型编号
                    ⋮
                    ⋮
nnToolKit1->lmsimu(1,&RetValue,ModelNo,ParaIn1,ParaIn2,InputFunc,OutputFunc,FileDir)；
        RetDouble=RetValue；

        ShowMessage("仿真成功！")；
        btnShowResult->Enabled = true；
}
```

5.2.5 VC 调用 nnToolKit 神经网络工具包实现混合编程

在 VC 中调用 nnToolKit 神经网络工具包实现混合编程，主要分为以下步骤：

1．创建 MFC 工程

打开 Viusal C++，新建一个 MFC AppWizard[exe]单文档工程 lm。在“操作”菜单中增加两个菜单项“训练”和“仿真”，同时在工程中相应增加两个对话框，类名为 CSDlg 和 CTDlg，分别对应“训练”和“仿真”菜单项。界面设计如图 5-13、5-14 所示。

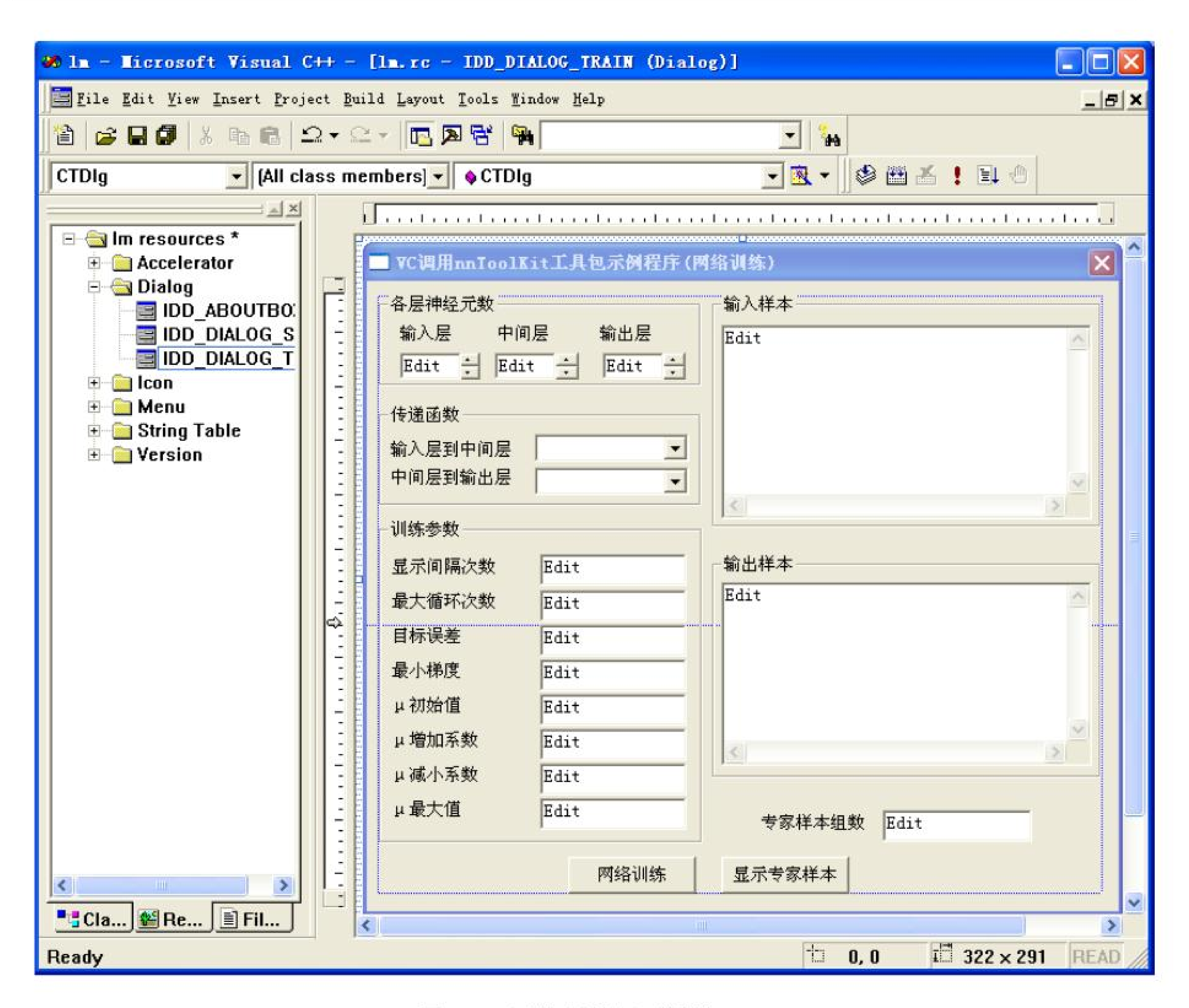


图 5-13 网络训练界面设计

网络仿真对话框设计如下：

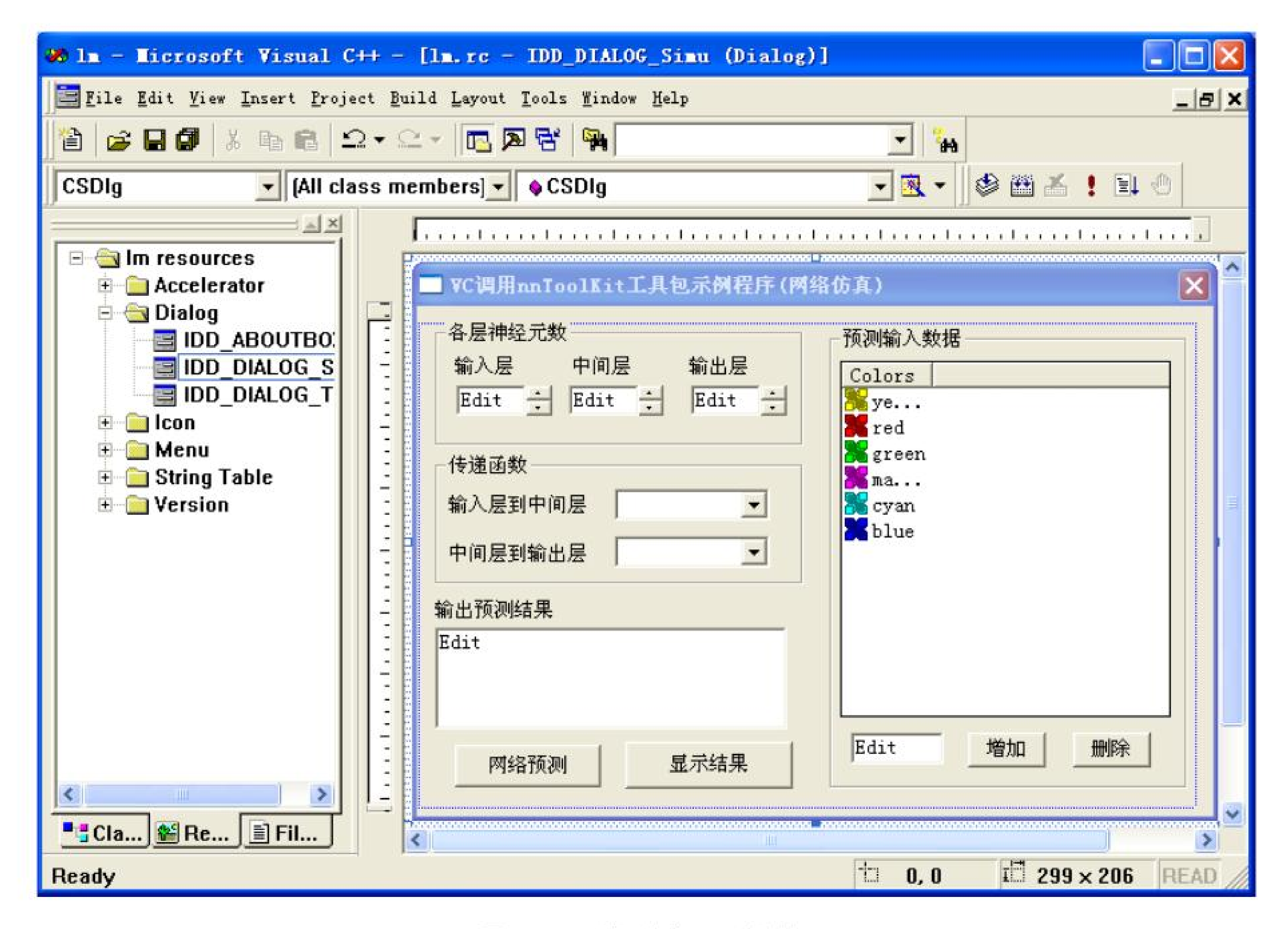


图 5-14 网络仿真界面设计

2．将 nnToolKit 动态库导入 VC

使用 MFC ClassWizard，添加一个新类，选择 From a type of library，在弹出的对话框中选中 nnToolKit.dll，如图 5-15 所示。

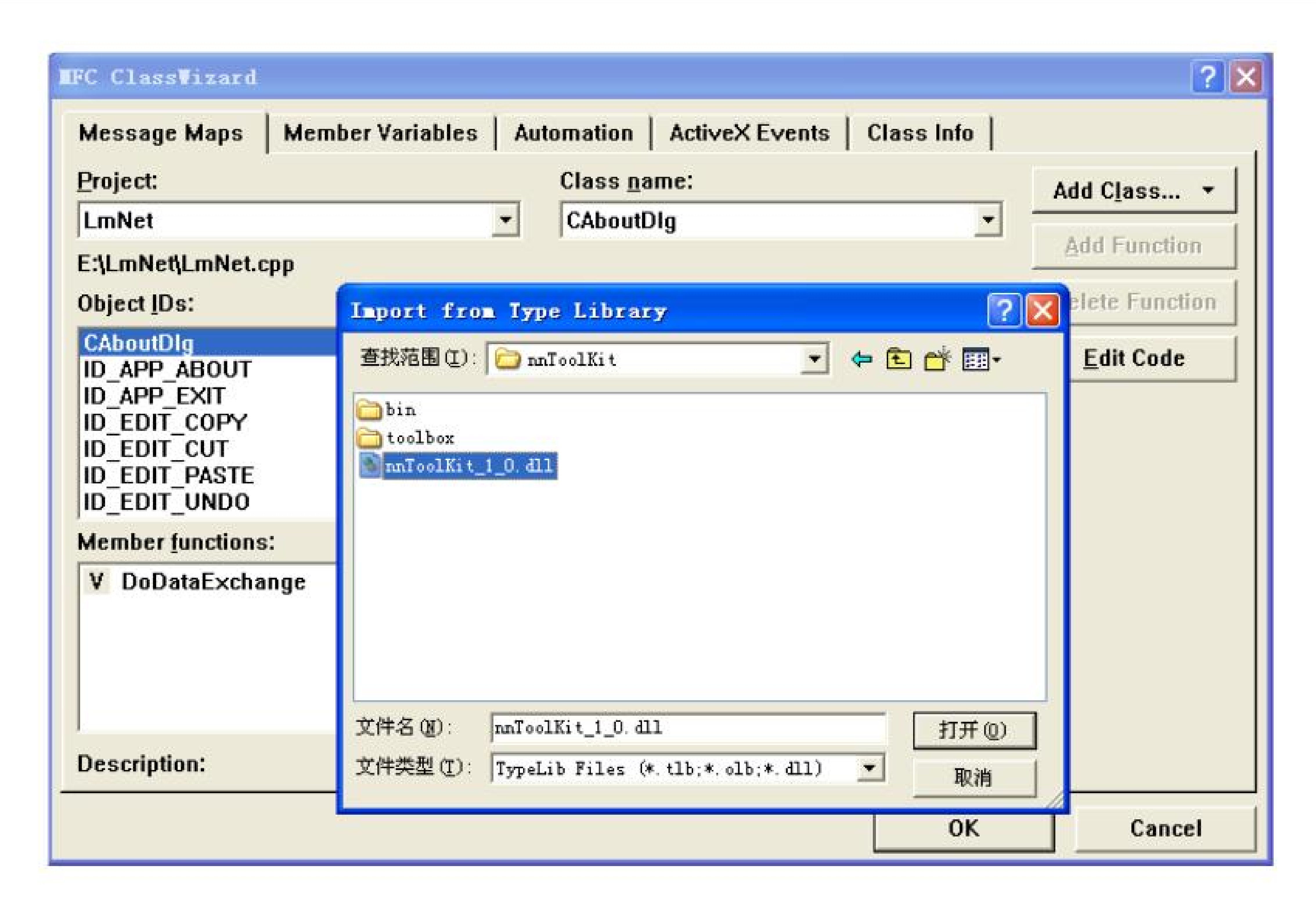


图 5-15 VC 中导入 nnToolKit 动态库

确定后，在 workspace 中可以发现这时自动产生了类 InnToolKit，如图 5-16 所示。

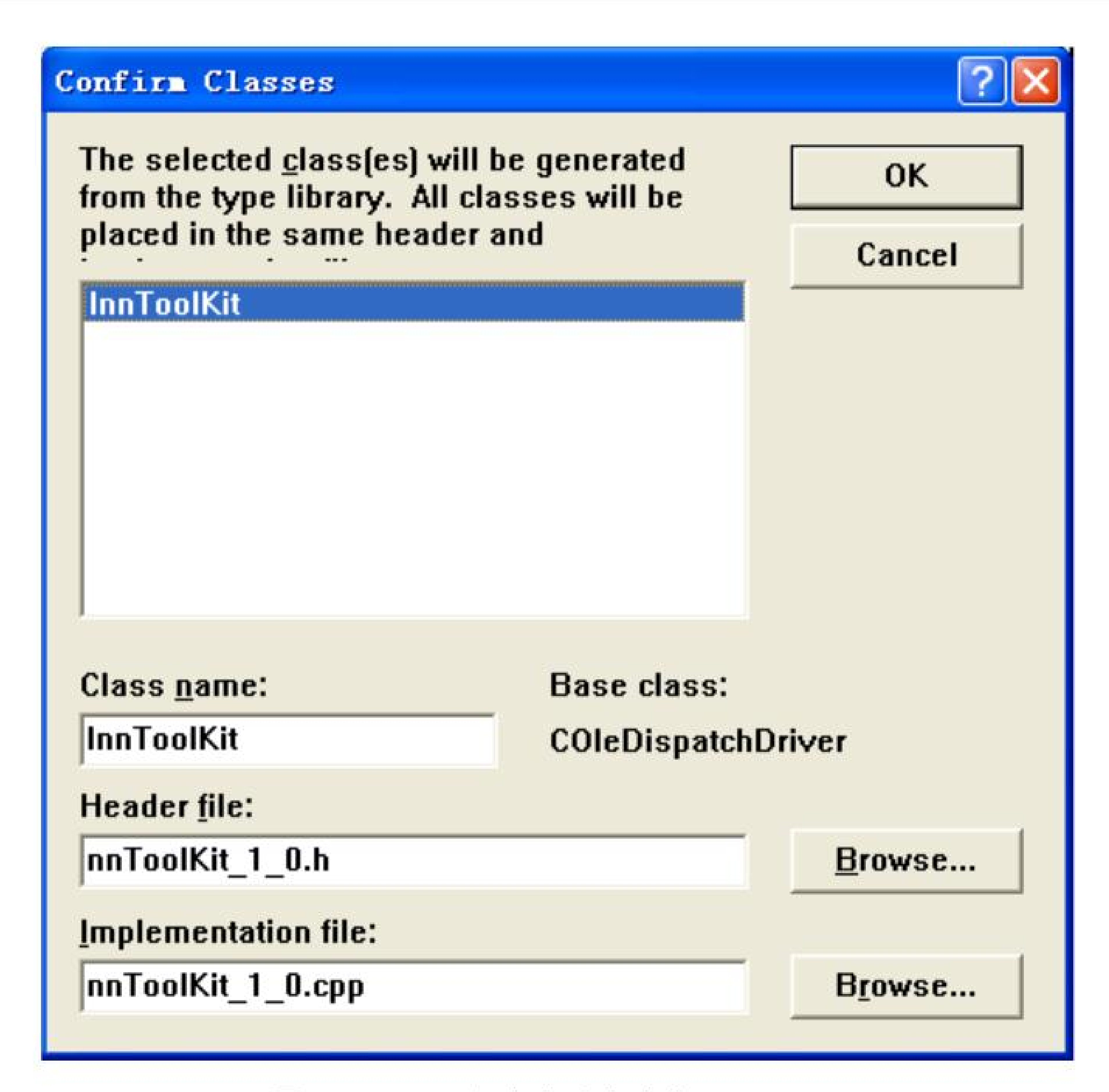


图 5-16 VC 工程中自动产生类 InnToolKit

3．代码实现

在 CSDlg 类中添加一个 nntoolkit 变量：

```
InnToolKit nn;
```

并在 SDlg.h 文件中添加：

```
#include "nntoolkit_1_0.h"
```

初始化 COM 组件的代码如下：

```
if(!nn.CreateDispatch("nnToolKit.nnToolKit.1_0"))
{
    MessageBox("加载组件出错!");
```

```
        exit(-1);
    }
  //调用 COM 件中的方法
  nn.lmsimu(1,&ret,ModelNo,np,sp,infun,outfun,dirname);
  //释放 COM 组件
  nn.ReleaseDispatch();
  //使用计算后返回的结果
  double result=ret.dblVal;
  m_outpre.Format("%f",result);
  MessageBox("仿真成功");

//网络训练部分的代码
void CTDlg::OnTrain()
{
   //获取网络参数
   UpdateData(TRUE);
   double n[4];
   n[0]=m_numinputt;
   n[2]=m_nummiddlet;
   n[1]=m_numoutputt;
   n[3]=m_numdata;
   COleVariant np;
   np.vt=VT_R8|VT_ARRAY;
   SAFEARRAYBOUND npbound[1]={4,0};
   np.parray=SafeArrayCreate(VT_R8,1,npbound);
   np.parray->pvData=n;
   //获取训练参数
   double t[8];
   t[0]=atof(m_trainp1);
   t[1]=atof(m_trainp2);
   t[2]=atof(m_trainp3);
   t[3]=atof(m_trainp4);
   t[4]=atof(m_trainp5);
   t[5]=atof(m_trainp6);
   t[6]=atof(m_trainp7);
```

```
t[7]=atof(m_trainp8);
COleVariant tp;
tp.vt=VT_R8|VT_ARRAY;
SAFEARRAYBOUND tpbound[1]={8,0};
tp.parray=SafeArrayCreate(VT_R8,1,tpbound);
tp.parray->pvData=t;
 //函数类型
 int curitem=m_funint.GetCurSel();
 CString funin;
 if(curitem==0)
    funin="logsig";
 else if(curitem==1)
    funin="tansig";
 else if(curitem==2)
    funin="purelin";
 COleVariant infun=funin;

 CString funout;
 curitem=m_funoutt.GetCurSel();
 if(curitem==0)
    funout="tansig";
 else if(curitem==1)
    funout="purelin";
 else if(curitem==2)
    funout="logsig";
 COleVariant outfun=funout;
 //模型名和目录
 COleVariant ModelNo="1";
COleVariant dirname=dir;
//输出设置
COleVariant ret;
//调用 nnToolKit
CSDlg d;
     if(!d.nn.CreateDispatch("nnToolKit.nnToolKit.1_0"))
{
```

```
        MessageBox("加载组件出错!");
        exit(-1);
    }
    d.nn.lmtrain(1,&ret,ModelNo,np,tp,infun,outfun,dirname);
    d.nn.ReleaseDispatch();
    double result=ret.dblVal;
    CString str;
    str.Format("网络训练成功，训练%f 次",result);
    MessageBox(str);

}
//网络仿真部分的代码
void CSDlg::OnSimu()
{
    // TODO: Add your command handler code here
    //输入输出节点数；
    double n[3];
    n[0]=m_numinput;
    n[1]=m_numoutput;
    n[2]=m_nummiddle;
    COleVariant np;
    np.vt=VT_R8|VT_ARRAY;
    SAFEARRAYBOUND npbound[1]={3,0};
    np.parray=SafeArrayCreate(VT_R8,1,npbound);
    np.parray->pvData=n;
    //函数类型
    int curitem=m_funin.GetCurSel();
    CString funin;
    if(curitem==0)
        funin="logsig";
    else if(curitem==1)
        funin="tansig";
    else if(curitem==2)
        funin="purelin";
    COleVariant infun=funin;
```

```
  CString funout;
  curitem=m_funout.GetCurSel();
  if(curitem==0)
     funout="tansig";
  else if(curitem==1)
     funout="purelin";
  else if(curitem==2)
     funout="logsig";
  COleVariant outfun=funout;
  //模型名和目录
  COleVariant ModelNo="1";
CString dir;
GetModuleFileName(NULL,dir.GetBuffer(100),100);
dir.ReleaseBuffer();
dir.TrimRight();
dir=dir.Left(dir.GetLength()-6)+"demo";

COleVariant dirname=dir;
  //预测数据
  //检测长度
  int numpre=m_list.GetItemCount();
  if(numpre!=m_numinput)
    { MessageBox("输入预测数据的个数不对，确认");return;}
  double s[7];
  for(int i=0;i<numpre;i++)
     s[i]=atof(m_list.GetItemText(i,1));
  COleVariant sp;
  sp.vt=VT_R8|VT_ARRAY;
  SAFEARRAYBOUND spbound[1]={7,0};
  sp.parray=SafeArrayCreate(VT_R8,1,spbound);
  sp.parray->pvData=s;
  //输出设置
  COleVariant ret;
  //调用 NNTOOLKIT
```

```
 if(!nn.CreateDispatch("nnToolKit.nnToolKit.1_0"))
 {
      MessageBox("加载组件出错!");
      exit(-1);
 }
 nn.lmsimu(1,&ret,ModelNo,np,sp,infun,outfun,dirname);
 nn.ReleaseDispatch();
 double result=ret.dblVal;
 m_outpre.Format("%f",result);
 MessageBox("仿真成功");
}
```

5.3 Excel 生成器（Excel Builder）

MATLAB 还提供了一个称为 Excel 生成器的工具，利用该工具，可以生成 DLL 组件和 VBA 代码。利用 DLL 组件，可以进行与前面 COM 生成器组件相似的操作。VBA 代码可以在 Excel 的 Visual Basic 编辑器中直接使用，可以保存为插件（Add-Ins）。

Excel 生成器创建的 COM 对象暴露给 VB 程序环境一个类，该类包含一系列称为方法的函数，对应于包含在组件工程中的原始 MATLAB 函数。本章主要介绍 MATLAB 与 Excel 的接口，并通过 Excel 生成器实现它们之间的接口。

5.3.1 创建 nnxToolKit 的 Excel 插件

用 MATLAB Excel 生成器创建 Excel 插件，主要包括 5 个步骤，即创建工程、管理 M 文件和 MEX 文件、生成组件、测试 VBA 模块、打包和分发组件。

1．创建 nnxToolKit 工程

在 MATLAB 命令行中输入命令 mxltool，打开 MATLAB Excel 生成器主窗口，它是 MATLAB Excel 生成器的主要工作环境。

在窗口中依次选择 File→New Project，打开“New Project Setting”对话框，如图 5-17 所示。

（1）在“Component name”文本框中输入组件名 nnxToolKit，输入组件名后，生成器会自动在“Class name”文本框中输入一个与组件名相同的类名，这里采用默认类名。

注意：尽管组件名与类名可以相同，但组件名不能与后面添加的 M 文件或 MEX 文件的名字相

同。

（2）在“Project Version”文本框中输入组件的版本号，默认版本号为 1.0。

（3）在“Project Directory”文本框中输入工程目录。工程目录指定编译和打包模型时，将工程和生成的文件放在那里，工程目录根据当前目录名和组件名自动创建。这里，将工程保存在目录“D:\MATLAB6p5\work\nnxToolKit”。

（4）在“Compile code in”方框中选择生成 C 代码还是 C++代码，用 C 代码写成的组件表现更好，而 C++组件可读性更，更便于修改，在此选择生成 C 代码。

（5）在“Compiler options”方框中可以创建一个编译模型的调试版本，当激活 MATLAB 编译器时，可以指定详细的输出，组件的调试版本具有下面的特点：

1) 允许反馈 通过反馈，所有出错报告显示 M 文件和发生错误的行。

2）允许使用 Visual Studio 调试器进行调试

在“New Project Settings”对话框上单击“OK”按钮，系统自动接受这些设置，它们成为工程的一部分，并且与后面添加到工程中的 M 文件或 MEX 文件的文件名一起被保存到工程中，这样，一个名为 nnxToolKit.mxl 的工程文件被自动保存到工程路径中。

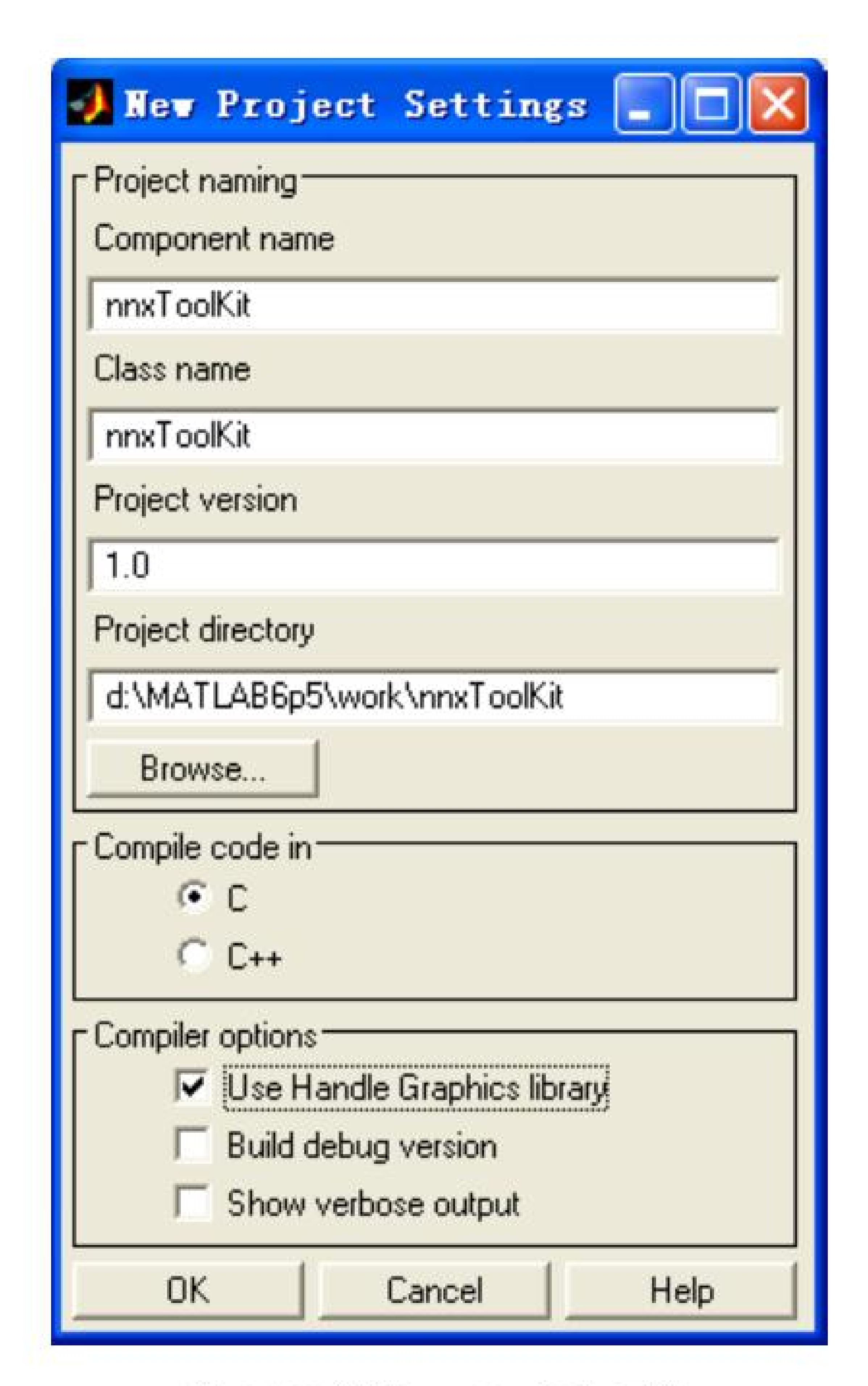


图 5-17 创建 nnxToolKit 工程

2．管理 M 文件和 MEX 文件（神经网络相关函数）

创建工程后，主窗口的“Project”,“Build”和“Component”等菜单选项变为可用，单击“Add File”按钮或从“Project”菜单中选择“Add File…”选项，在工程中将已调试好的神经网络相关函数（LmTrain.m、LmSimu.m 等）加入到项目中。

使用“Remove”按钮或从“Project”菜单中选择“Remove File…”选项，将删除所有已经选定的 M 文件或 MEX 文件。单击“Edit”按钮或从“Project”菜单中选择“Edit File…”选项，或双击 M 文件的文件名，将在 MATLAB 编辑器中打开选定的 M 文件，并可以进行修改和调试。

3．生成 nnxToolKit 组件

定义好工程设置并添加了相关神经网络函数后，通过“Build”菜单中的“Excel/COM Files”选项或直接单击“Build”按钮来调用 MATLAB 编译器，生成一个可配置的 DLL 文件和必要的 VBA

代码。同时中间源文件写到 D:\MATLAB6p5\work\nnxToolKit\src 目录中，将进行配置的输出文件（DLL 文件和 VBA 文件）写到“D:\MATLAB6p5\work\nnxToolKit\distrib”目录中，“Build Status”面板显示生成过程的输出信息，如图 5-18 所示。

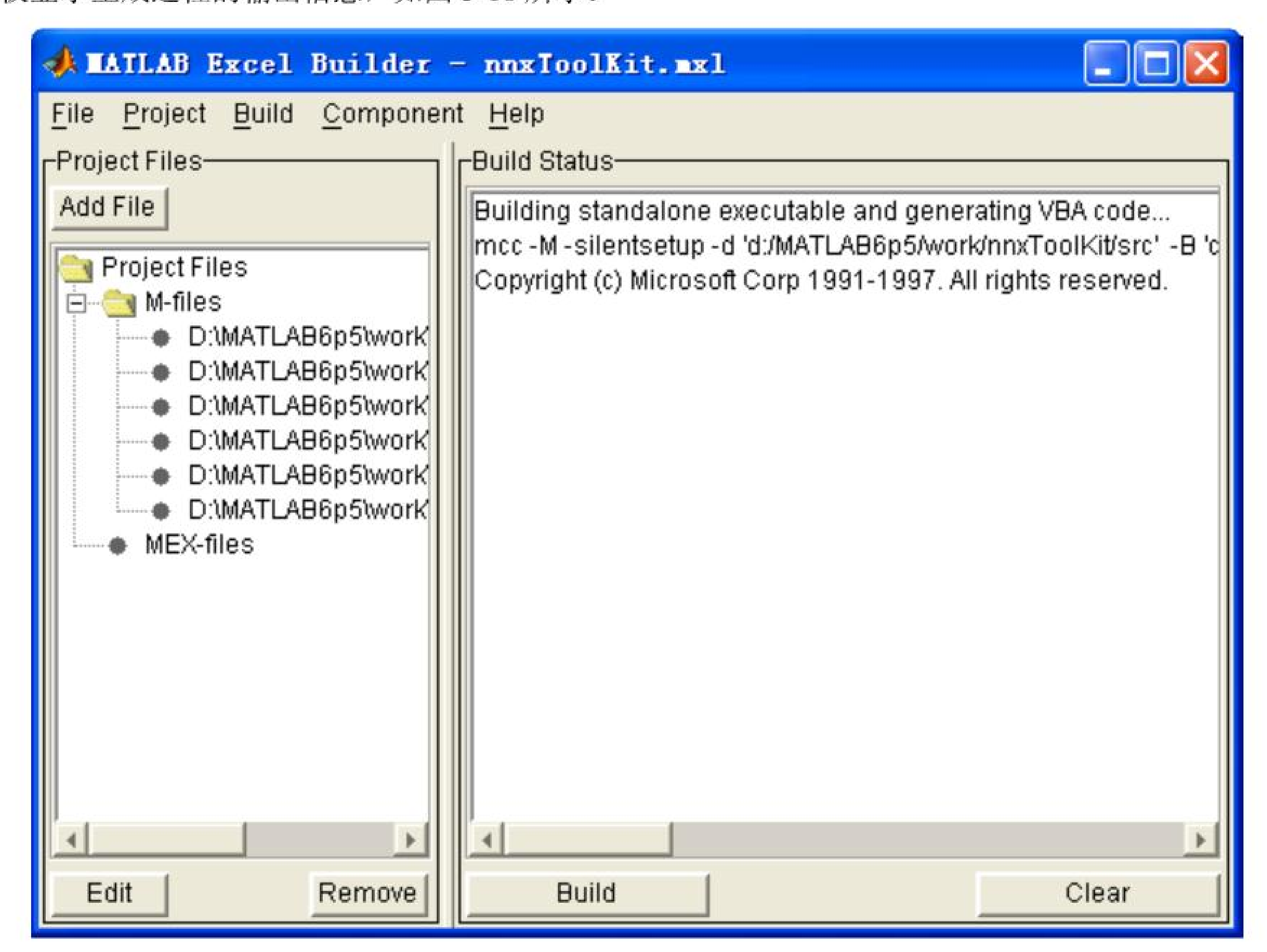


图 5-18 COM 生成器主界面

同时，在“D:\MATLAB6p5\work\nnxToolKit\build.log”文件中，记录了生成过程的输出信息。通过阅读该 Log 文件，用户可以对编译过程中的出错信息进行跟踪。

4．打包和发布组件

如果模块编译成功，并且创建了 Excel 插件以后，就可以准备包装组件，把它发布给终端用户。从“Component”菜单中选择“Package Component”选项，将创建表 5-3 所示的系列文件：

表 5-3 自解压文件 nnxToolKit.exe 包含的文件

文　件	功　能

_install.bat	由自解压可执行程序运行的脚本
nnxToolKit_1_0.dll	编译后的组件
mglinstaller.exe	数学库和图形库安装器
mwcomutil.dll	生成器工具库
mwregsvr.exe	在目标机器上注册 DLL 可执行程序
*.xla	任何在目录中找到的插件文件

自解压可执行程序的名称为 nnxToolKit.exe。在计算机上运行安装器，将按以下步骤进行：

（1）mglinstaller 安装 MATLAB C/C++数学库和图形库。

（2）添加<application>\bin\win32 目录，其中<application>表示配置应用的根目录。

（3）mwregsvr 注册 mwcomutil.dll 和 nnxToolKit_1_0.dll。

（4）mglinstaller 将所有 Excel 插件（*.xla）写到当前目录位置。

（5）要使用 Excel 插件，启动 Excel，选择“工具”→“加载宏…”，并选择 LmNet.xla 文件。

必须在每个需要安装组件的计算机上重复本发布过程，下面介绍组件的安装和使用。

5.3.2 nnxToolKit 组件安装

直接运行 nnxToolKit.exe 安装文件，程序首先完成自解压，然后进入安装过程，具体步骤类似于 nnTookKit 的安装，在此不再另述，读者可参见 5.2.2 章节介绍。

5.3.3 将 nnxToolKit 组件集成到 VBA 中

nnxToolKit 组件建好以后，通过实现必要的 VBA 代码后，可将它集成到 Excel 中去。按照下面的步骤打开 Excel 并选择必要的库文件开发插件。

1．启动 Excel。

2．从 Excel 主菜单中，按顺序选择“工具”→“宏”→“Visual Basic 编辑器”，启动 Excel 的 Visual Basic 编辑器。

3．在编辑器的“工具”菜单中选择“引用…”选项，显示“引用”对话框。在该对话框中选择“nnxToolKit 1.0 Type Library”和“MWComUtil 1.0 Type Library”。

插件需要一些初始化代码和一些全局变量来控制应用程序的状态。要达到这个目的，按照后面的步骤实现 Visual Basic 代码模块。

4．在工程窗口中右击 VBAProject 选项并从弹出式菜单中按顺序选择“插入”→“模块”。

5．在 VBAProject 中的 Modules 下面出现一个新的模块。在该模块的属性页中，将 Name 属性设置为 LmNetMain，如图 5-19 所示。

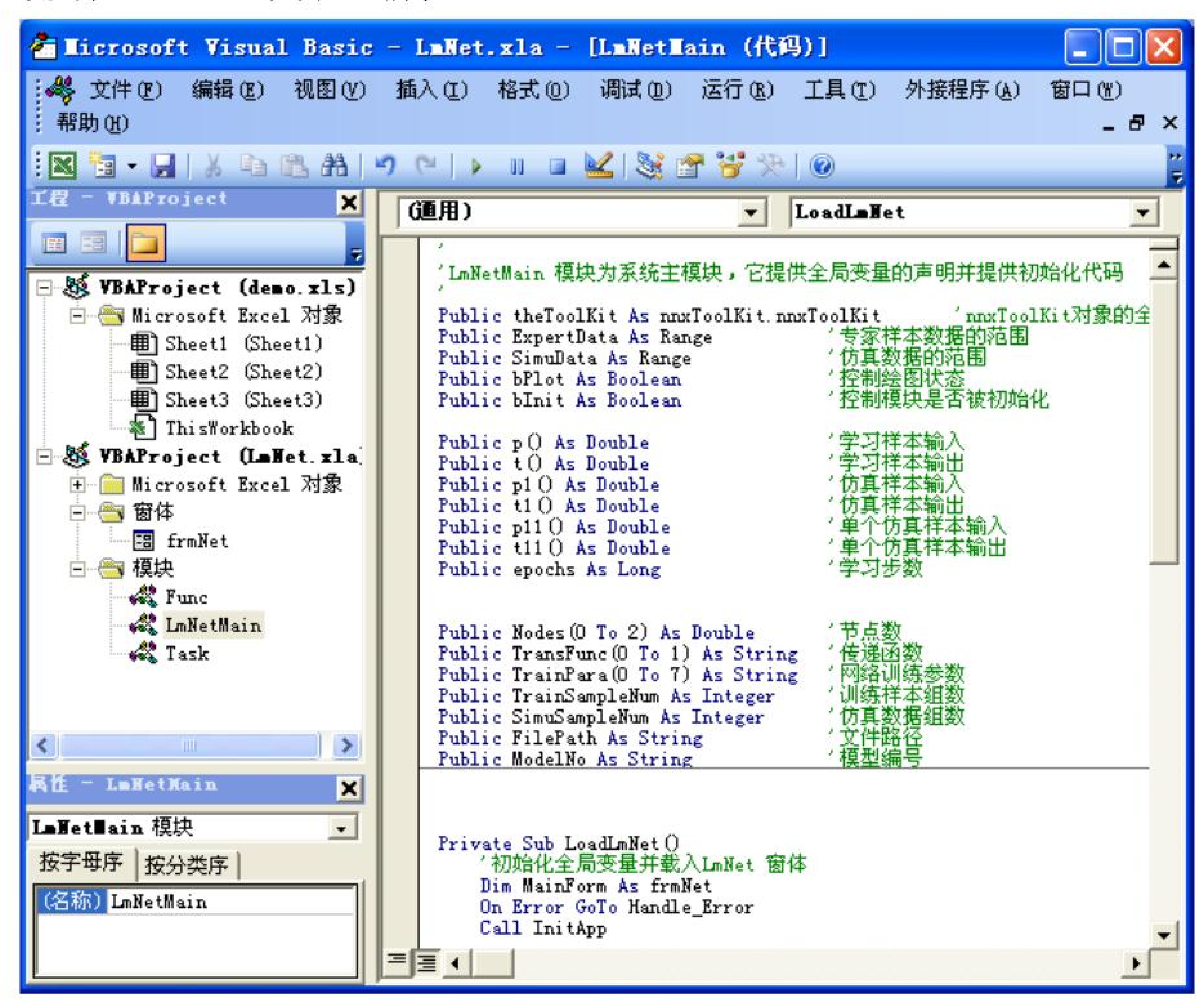


图 5-19 在 VBA 工程中插入模块

在 LmNetMain 模块中输入下面的代码：

```
'
'LmNetMain 模块为系统主模块，它提供全局变量的声明并提供初始化代码
'
Public theToolKit As nnxToolKit.nnxToolKit    'nnxToolKit 对象的全局实例
```

```
Public ExpertData As Range                '专家样本数据的范围
Public SimuData As Range                  '仿真数据的范围
Public bPlot As Boolean                   '控制绘图状态
Public bInit As Boolean                   '控制模块是否被初始化

Public p() As Double                      '学习样本输入
Public t() As Double                      '学习样本输出
Public p1() As Double                     '仿真样本输入
Public t1() As Double                     '仿真样本输出
Public p11() As Double                    '单个仿真样本输入
Public t11() As Double                    '单个仿真样本输出
Public epochs As Long                     '学习步数

Public Nodes(0 To 2) As Double            '节点数
Public TransFunc(0 To 1) As String        '传递函数
Public TrainPara(0 To 7) As String        '网络训练参数
Public TrainSampleNum As Integer          '训练样本组数
Public SimuSampleNum As Integer           '仿真数据组数
Public FilePath As String                 '文件路径
Public ModelNo As String                  '模型编号

Private Sub LoadLmNet()
    '初始化全局变量并载入 LmNet 窗体
    Dim MainForm As frmNet
    On Error GoTo Handle_Error
    Call InitApp
    Set MainForm = New frmNet
    Call MainForm.Show
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub

Private Sub InitApp()
'初始化类和库
```

```
'运行一次当前的 Excel 进程
If bInit Then Exit Sub
    On Error GoTo Handle_Error
    If theToolKit Is Nothing Then
        Set theToolKit = New nnxToolKit.nnxToolKit
    End If
    bInit = True
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
'
'打开已建网络模型
'
Sub OpenLmNet()
    ModelName = Application.GetOpenFilename(filefilter:="Excel workbooks,*.xls", Title:="
打开 LmNet 网络模型")
    If (ModelName = False) Then
        Exit Sub
    End If
End Sub
'
'新建网络模型
'
Sub NewLmNet()
    ModelName = Application.NewWorkbook
    If (ModelName = False) Then
        Exit Sub
    End If
End Sub
```

5.3.4 创建图形用户界面

集成处理的下一步是用 Visual Basic 编辑器为 nnxToolKit 插件开发一个神经网络应用。按照以

下步骤创建一个新的用户窗体，并在窗体上添加必要的控件。

1．在工程窗口中右击 VBAProject 选项，并从弹出式菜单中按顺序选择“插入”→“用户窗体”选项。

2．现在，在 VBA 工程中，一个新的窗体显示在 Forms 下面。在窗体的属性页中，将 Name 属性设置为 frmLmNet，将 Caption 属性设置为：VBA 调用 nnxToolKit 示例。

3．给空白的窗体中添加控件，如图 5-20 所示，并按表 5-4 中的内容设置控件属性。



图 5-20 创建 frmLmNet 窗体

表 5-4 属性设置

控件类型	控件名称	属　　性
Frame	Frame1	Caption = 网络各层节点数
TextBox	txtNodeNum1	注：输入层节点数
TextBox	txtNodeNum2	注：中间层节点数
TextBox	txtNodeNum3	注：输出层节点数
Frame	Frame2	Caption = 传递函数
CombBox	cmbFunc1	注：输入层至中间层传递函数
CombBox	cmbFunc2	注：中间层至输出层传递函数
Frame	Frame3	Caption = 网络参数
TextBox	txtNetPara1	注：显示间隔次数
TextBox	txtNetPara2	注：最大循环次数
TextBox	txtNetPara3	注：目标误差
TextBox	txtNetPara4	注：动量常数
TextBox	txtNetPara5	注：学习速率
TextBox	txtNetPara6	注：最大误差比例
Frame	Frame4	Caption = 数据区域
RefEdit	refedtExpertIn	注：训练样本输入数据区域
RefEdit	refedtExpertOut	注：训练样本输出数据区域
RefEdit	refedtSimuIn	注：测试样本输入数据区域
RefEdit	refedtSimuOut	注：仿真结果输出数据区域

窗体和控件创建完毕以后，右击窗体并从弹出式菜单中选择“查看代码”选项。然后在代码窗口中输入下面的代码。

'

```
'frmNet 窗体激活事件
'
Private Sub UserForm_Activate()
'UserForm 被激活时的事件句柄。本函数在显示窗体以前被调用'
'并用保存在全局变量中的值初始化所有控制
On Error GoTo Handle_Error
    cmbFunc1.AddItem "purelin", 0
    cmbFunc1.AddItem "tansig", 1
    cmbFunc1.Text = "purelin"

    cmbFunc2.AddItem "purelin", 0
    cmbFunc2.AddItem "tansig", 1
    cmbFunc2.Text = "tansig"

    '切换网络节点参数到建模
    txtNodeNum1.Text = ActiveWorkbook.Sheets("Sheet1").Cells(3, 2)
    txtNodeNum2.Text = ActiveWorkbook.Sheets("Sheet1").Cells(4, 2)
    txtNodeNum3.Text = ActiveWorkbook.Sheets("Sheet1").Cells(5, 2)

    '切换网络训练参数到建模
    txtNetPara1.Text = ActiveWorkbook.Sheets("Sheet1").Cells(3, 4)
    txtNetPara2.Text = ActiveWorkbook.Sheets("Sheet1").Cells(4, 4)
    txtNetPara3.Text = ActiveWorkbook.Sheets("Sheet1").Cells(5, 4)
    txtNetPara4.Text = ActiveWorkbook.Sheets("Sheet1").Cells(6, 4)
    txtNetPara5.Text = ActiveWorkbook.Sheets("Sheet1").Cells(7, 4)
    txtNetPara6.Text = ActiveWorkbook.Sheets("Sheet1").Cells(8, 4)

    '切换数据显示区域到建模
    refedtExpertIn.Text = ActiveWorkbook.Sheets("Sheet1").Cells(3, 6)
    refedtExpertOut.Text = ActiveWorkbook.Sheets("Sheet1").Cells(4, 6)
    refedtSimuIn.Text = ActiveWorkbook.Sheets("Sheet1").Cells(5, 6)
    refedtSimuOut.Text = ActiveWorkbook.Sheets("Sheet1").Cells(6, 6)

    If theToolKit Is Nothing Then Exit Sub
    '用当前值初始化控制
```

```
        If Not ExpertData Is Nothing Then
            refedtExpertData.Text = ExpertData.Address
        End If
        Exit Sub
Handle_Error:
        MsgBox (Err.Description)
End Sub

'模型切换到 Excel
'
Private Sub btnToExcel_Click()
        '切换网络节点参数到 Excel
        ActiveWorkbook.Sheets("Sheet1").Cells(3, 2) = txtNodeNum1.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(4, 2) = txtNodeNum2.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(5, 2) = txtNodeNum3.Text

        '切换网络训练参数到 Excel
        ActiveWorkbook.Sheets("Sheet1").Cells(3, 4) = txtNetPara1.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(4, 4) = txtNetPara2.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(5, 4) = txtNetPara3.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(6, 4) = txtNetPara4.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(7, 4) = txtNetPara5.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(8, 4) = txtNetPara6.Text

        '切换数据显示区域到 Excel
        ActiveWorkbook.Sheets("Sheet1").Cells(3, 6) = refedtExpertIn.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(4, 6) = refedtExpertOut.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(5, 6) = refedtSimuIn.Text
        ActiveWorkbook.Sheets("Sheet1").Cells(6, 6) = refedtSimuOut.Text
        Unload Me
End Sub

Private Sub btnTrain_Click()
        'ReDim p(SampleNum, Node(0)) As Double
        'ReDim t(SampleNum, Node(2)) As Double
```

```
    'p = Range(refedtExpertIn.Text)
    't = Range(refedtExpertOut.Text)
    Call NetModel_Init
    On Error GoTo Handle_Error
    '返回值
    Dim retstr As Variant
    '网络训练基本参数
    Dim NetTPara(4) As Double
    '模型编号
    Dim vModelNo As Variant
    vModelNo = ModelNo
    For i = 0 To 2
        NetTPara(i) = Nodes(i)
    Next
    NetTPara(3) = TrainSampleNum    '样本个数
    vModelNo = 1
    '网络训练调用参数说明
    '1，一个返回参数
    'retstr，返回值
    '网络模型基本参数，Variant
    '神经网络参数，Double，包括输入层节点数；输出层节点数；中间层节点数；网络训练样本个数。
    '神经网络训练参数，若为"-1"表示选择默认参数
    '网络仿真时输入层至中间层的传递函数
    '网络仿真时中间层至输出层的传递函数
    '程序运行时的当前目录
    Call theToolKit.lmtrain(1, retstr, vModelNo, NetTPara, -1, TransFunc(0), TransFunc(1), FilePath)
    MsgBox ("训练完成，您现在可以进行仿真操作了！")
Handle_Error:
  MsgBox (Err.Description)
End Sub
'
'神经网络仿真
'
```

```
Private Sub btnSimu_Click()
    '神经网络仿真参数
    Dim SimulatePara(7) As Double
    '模型编号
    Dim ModelNo As String
    ModelNo = "1"
    On Error GoTo Handle_Error

    'For i = 0 To Nodes(0) - 1
    '  SimulatePara(i) = Trim(Left(DataLine，  5))
    '  DataLength = Len(DataLine)
    '  DataLine = Trim(Right(DataLine，  DataLength - 5))

    'Next
    '调试用数据
    SimulatePara(0) = 0.9
    SimulatePara(1) = 0.9
    SimulatePara(2) = 0.063
    SimulatePara(3) = 0.9
    SimulatePara(4) = 0.025
    SimulatePara(5) = 0.467
    SimulatePara(6) = 0.9

    '网络仿真调用参数说明
    '1，一个返回参数
    'retstr，返回值
    '网络模型基本参数，Variant
    '神经网络参数，Double，包括输入层节点数；输出层节点数；中间层节点数
    '神经网络仿真参数，若为"-1"表示选择默认参数
    '网络仿真时输入层至中间层的传递函数
    '网络仿真时中间层至输出层的传递函数
    '程序运行时的当前目录
    Call theToolKit.lmsimu(1, retstr, ModelNo, Nodes, SimulatePara, TransFunc(0),
TransFunc(1), FilePath)
    MsgBox ("仿真成功，点击'数据/显示测试结果'查看仿真结果！")
```

```
        Exit Sub
    Handle_Error:
        MsgBox (Err.Description)
    End Sub
```

集成的最后一步是给 Excel 添加一个菜单选项。这样，就可以从 Excel 的“工具”菜单中调用该工具了。进行这一步工作，需要为工作簿的 AddinInstall 和 AddinUninstall 事件添加事件句柄。该菜单项调用 LoadLmNet 模块中的 LoadLmNet 函数。按照下面的步骤实现菜单选项：

1．在 Visual Basic 工程窗口中右击“ThisWorkbook”选项，并从弹出式菜单中选择“查看代码”选项，结果如图 5-21 所示。

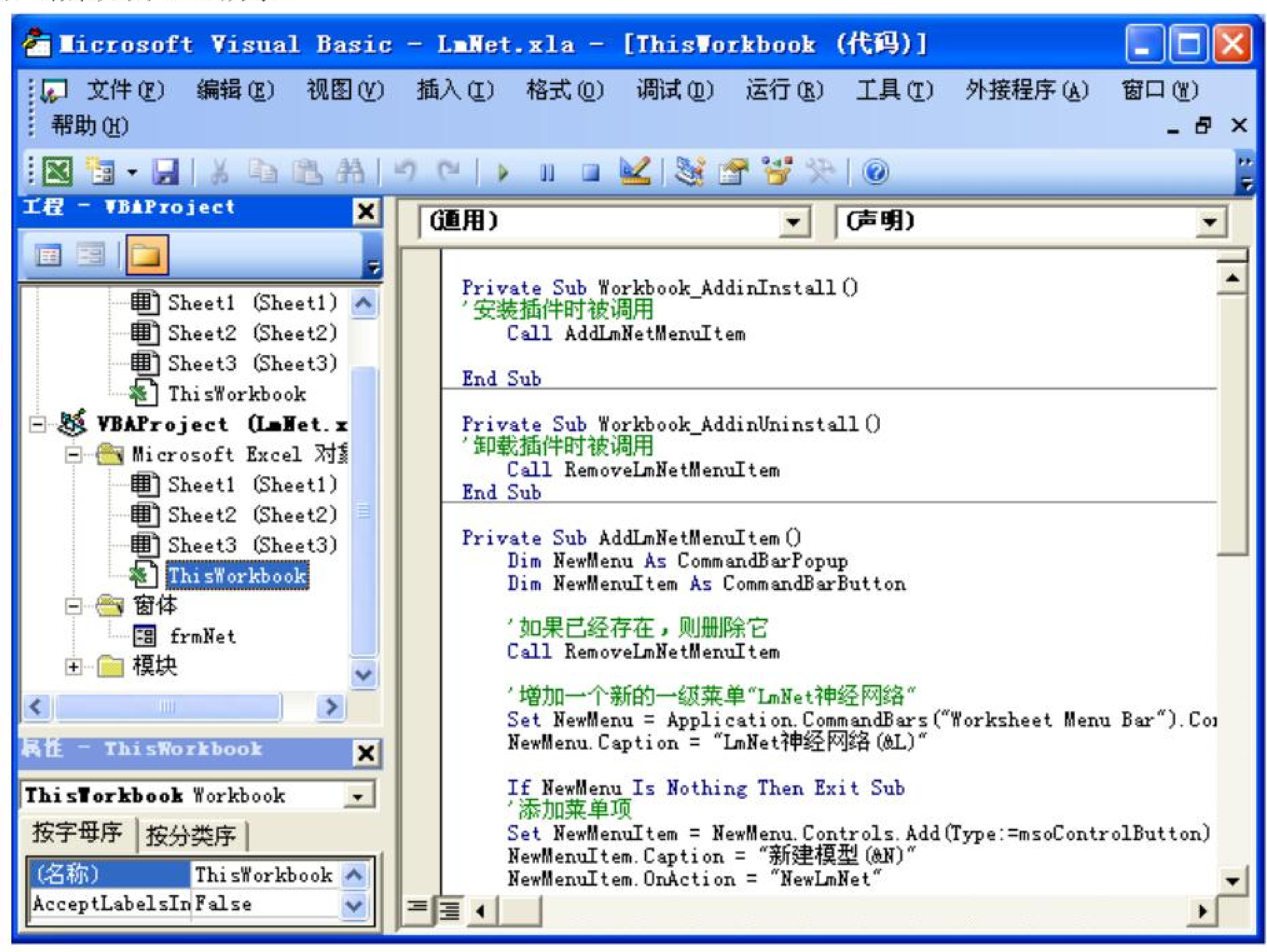


图 5-21　与 Excel 集成

2．将下面的代码放到 ThisWorkbook 对象中。

```
'
'安装插件时被调用
'
Private Sub Workbook_AddinInstall()
    Call AddLmNetMenuItem
End Sub
'
'卸载插件时被调用
'
Private Sub Workbook_AddinUninstall()
    Call RemoveLmNetMenuItem
End Sub
'
'在 Excel 中加载菜单项
'
Private Sub AddLmNetMenuItem()
    Dim NewMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton
    '如果已经存在，则删除它
    Call RemoveLmNetMenuItem
    '增加一个新的一级菜单"LmNet 神经网络"
    Set NewMenu = Application.CommandBars("Worksheet Menu Bar").Controls.Add(msoControlPopup)
    NewMenu.Caption = "LmNet 神经网络(&L)"
    If NewMenu Is Nothing Then Exit Sub
    '添加菜单项
    Set NewMenuItem = NewMenu.Controls.Add(Type:=msoControlButton)
    NewMenuItem.Caption = "新建模型(&N)"
    NewMenuItem.OnAction = "NewLmNet"
    Set NewMenuItem = NewMenu.Controls.Add(Type:=msoControlButton)
    NewMenuItem.Caption = "打开模型(&O)"
    NewMenuItem.OnAction = "OpenLmNet"
    Set NewMenuItem = NewMenu.Controls.Add(Type:=msoControlButton)
```

```
        NewMenuItem.Caption = "切换至建模(&C)"
        NewMenuItem.OnAction = "LoadLmNet"

        Set NewMenuItem = NewMenu.Controls.Add(Type:=msoControlButton)
        NewMenuItem.Caption = "保存模型(&S)"
        NewMenuItem.OnAction = "SaveLmNet"
End Sub
'
'在 Excel 中移走菜单项
'
Private Sub RemoveLmNetMenuItem()
        Dim bar As CommandBar
        Dim Ctrl As CommandBarControl
        On Error Resume Next
        Set bar = Application.CommandBars(1)
       For Each bar In CommandBars
            If bar.Name = "Worksheet Menu Bar" Then
                For Each Item In bar.Controls
                    If Item.Caption = "LmNet 神经网络(&L)" Then
                        Item.Delete
                    End If
                Next Item
            End If
        Next bar
End Sub
```

其中，Workbook_AddinInstall 过程调用 AddLmNetMenuItem 过程，在窗体中添加菜单项，它在安装插件时被调用；Workbook_AddinUninstall 过程在卸载插件时被调用；AddLmNetMenuItem 过程在“Tools”菜单中添加一个子菜单项；RemoveLmNetMenuItem 过程把“VBA 调用 nnxToolKit 示例”菜单英从“Tools”菜单中删除。

5.3.5 保存和测试插件

1．将插件命名为“VBA 调用 nnxToolKit 示例”，按照下面的步骤保存它：

- 在 Excel 主菜单中依次选择“文件”→“属性”；

- 当“Workbook 属性”对话框出现时，选择“摘要信息”选项卡，并输入“VBA 调用 nnxToolKit 示例”作为工作簿的标题；
- 单击“OK”，保存编辑内容；
- 从 Excel 主菜单中依次选择“文件”→“另存为…”；
- 在“另存为”对话框中选择“Microsoft Excel Add-In(*.xla)”作为文件类型；
- 输入“LmNet.xla”作为文件名，并单击“保存”按钮，保存插件。

在分发插件以前，对它进行测试。本示例主要通过对指定的专家样本数据进行训练，然后再对指定的测试数据进行仿真。故本例需要将样本数据保存到 Excel 工作簿的工作区域中，对于训练过程误差和仿真结果，用图形表示。

2．通过下面的步骤完成示例程序：

- 启动一个新的 Excel 进程，它有一个空白的工作簿；
- 从菜单中依次选择“工具”→“加载…”，打开“加载宏”对话框；
- 在“加载宏”对话框中单击“浏览…”按钮；
- 找到 LmNet.xla 文件并单击“确定”按钮；
- 在列表框中找到“VBA 调用 nnxToolKit 示例”插件，选择它；
- 单击“OK”按钮，装载插件。

本插件在 Excel 的“工具”菜单下安装一个菜单项。可以通过依次选择“工具”→“VBA 调用 nnxToolKit 示例”来显示本神经网络应用的图形用户界面。后面章节将详细介绍通过调用此插件来创建基于 VBA 的神经网络应用程序。

5.3.6 分发应用程序

将 COM 组件和所有的支持库打包到一个自解压可执行程序 nnxToolKit.exe 中。这样，本包就可以安装到其他需要使用 nnxToolKit 组件的计算机上了。同样需要将 LmNet.xla 文件复制到所有的在 Excel 中使用本组件的计算机中。

5.3.7 应用示例

下面通过前面章节发布的应用程序，快速建立一个神经网络模型，用以实现对电力行业的漏窃电进行预测。

1．案例描述

窃电者为了达到窃电目的，往往采用各种窃电手法进行窃电，手法五花八门，但万变不离其宗，最常见的是从电能计量原理入手。一个电能表计量电量的多少，主要决定于电压、电流、功率因数三因素和时间的乘积，因此，只要想办法改变三要素中的任何一个要素都可以使电表慢转、停转甚

至反转，从而达到窃电的目的；另外，通过采用改变电表本身的结构性能的手法，使电表慢转，也可以达到窃电的目的。

为了有效地抑制这种窃电行为，为开展窃电侦查工作和计量装置检查工作提供依据和锁定范围。有必要通过归纳出若干指标，同时根据最终用电客户用电现场计量装置的报警数据，结合客户历史用电数据和相关线路的线损数据，总结出一个预测分析模型。以便为管理者提供漏窃电的嫌疑人列表和嫌疑系数。

2．案例分析

预测模型的建立，关键在于预测模型及评价指标的选取（注意：在选取指标时，要充分考虑到采集数据的可行性）及数据的可靠性，目前有一种比较常用的简便的预测分析方法（加权平均法），实现对漏窃电的预测，如图 5-22 所示。

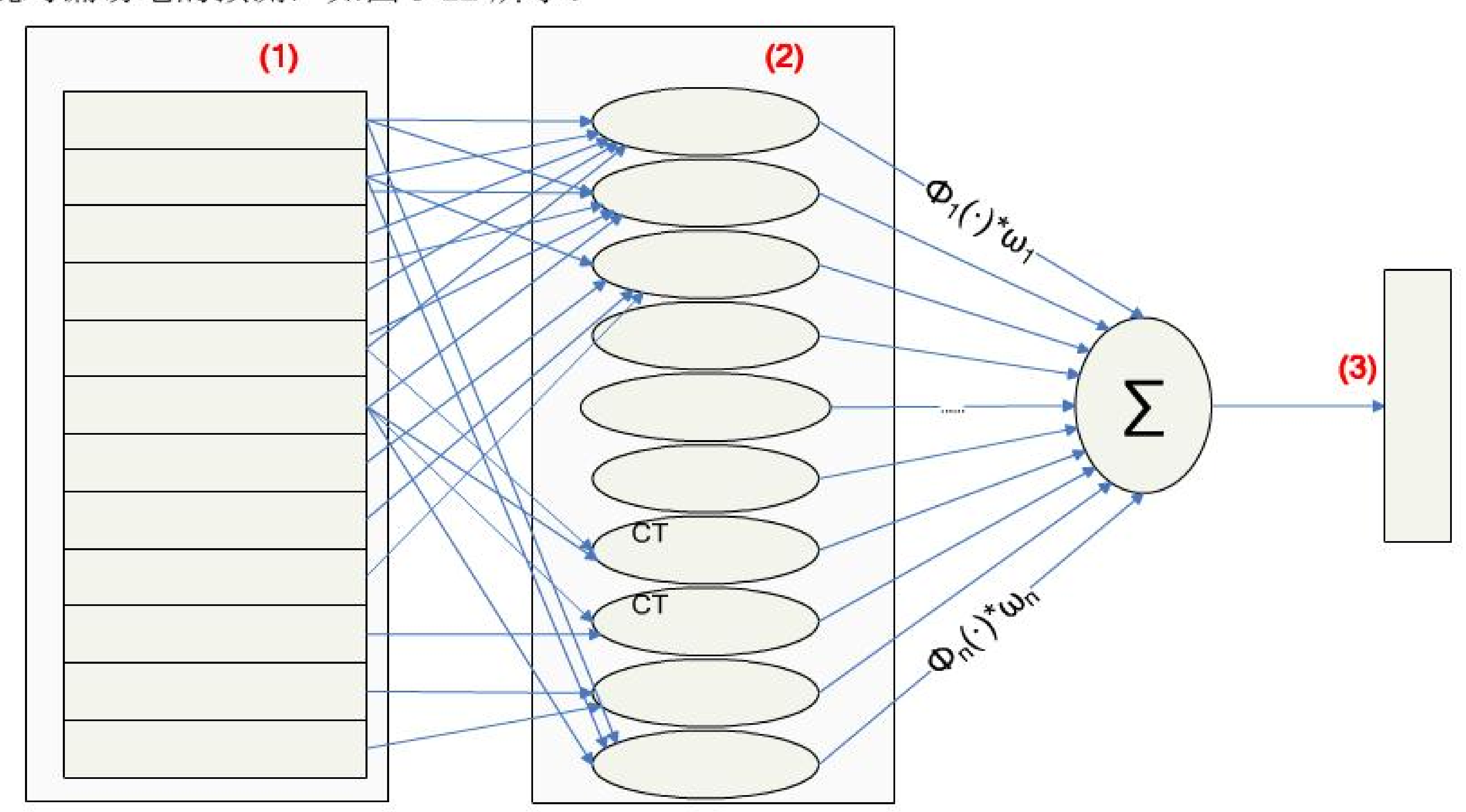


图 5-22 漏窃电预测之加权平均模型

这种基于线性加权模型的漏窃电预测方法的优点是数据容易采集，使用简单，但也存在很大的局限性，如影响漏窃电的各个预测指标之间有些相互影响，有些相互独立，呈现出复杂的非线性关系，用上述方法做出的评价比较粗糙。同时这种线性分析方法，仅从电压断相、电压缺相、示度下降等指标进行预测分析，显然其中间过程全部被忽略了，而且由于判断各指标的数据源部分又有明显交叉现象，因此有必要进行更深入的数据挖掘。即从电压断相、电压缺相、示度下降等指标的源头，通过判断各指标的数据源，并综合其它指标，如：线损率、用户单位产品耗电量、用户功率因数等指标，来重新确定预测分析模型的评价指标。本案例通过建立一个具有 10 个输入，一个输出的

神经网络预测模型，用以实现对漏窃电的预测。预测评价指标体系如图 5-23 所示。

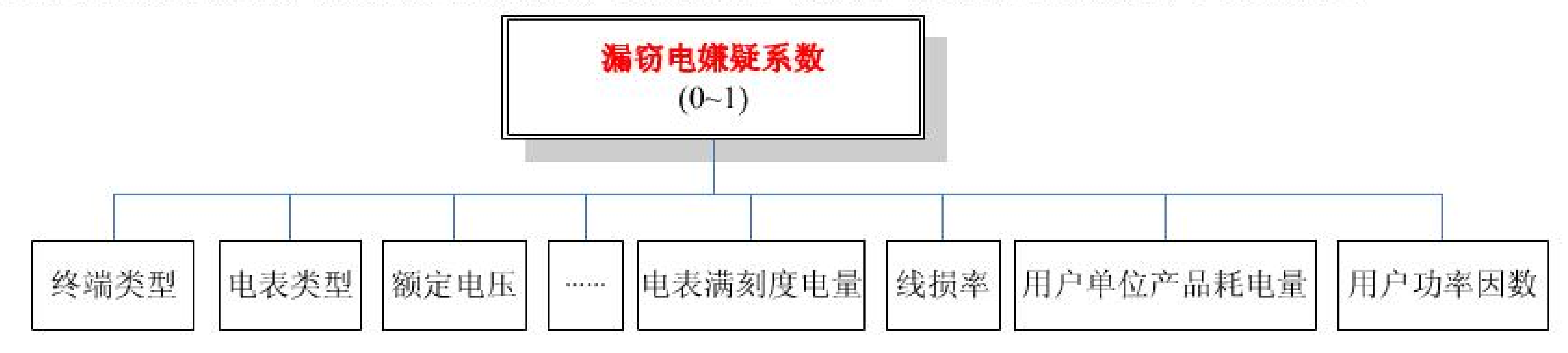


图 5-23 漏窃电预测评价指标

同时建立如表 5-5 所示的专家样本数据。

表 5-5 漏窃电预测专家样本数据

终端类型	电表类型	额定电压	……	线损率	用户功率因数	漏窃电嫌疑系数

3．程序实现

步骤如下：

⑴ 启动 Excel，在菜单中选择“工具”→“加载宏”→“浏览”，选定 LmNet.xla 文件并确定，在可用加载宏框中会出现宏“VBA 神经网络示例程序”，如图 5-24 所示，打勾后“确定”，这时在 Excel 菜单中将出现一个新的“LmNet 神经网络”菜单项。

(2) 在 Sheet1 表单项中输入神经网络建模的相关数据（如图 5-25 所示），在 Sheet2 表单项中输入专家样本数据和预测数据。



图 5-24 加载自定义神经网络宏



图 5-25 准备建模相关数据

(3) 网络建模相关数据准备好后，点击菜单“LmNet 神经网络”→“切换至建模”，将打开神经网络建模应用程序，如图 5-26。



图 5-26 模型训练和预测

通过系统提供的功能，可以快速地实现网络训练，模型训练完成后，就可以对指定的数据进行预测了。

练习题

1、除了本书介绍的混合编程方法外，你还了解哪些混合编程技术？各有何特点？

2、用 VB 调用 nnTooKit 神经网络工具包的方法，实现 4.5.2 节所示的地基沉降量预测。

第 6 章 神经网络混合编程案例

前面几章我们介绍了神经网络算法及混合编程方法，本章我们将基于实际应用，完整地介绍一个混合编程案例——个人资信评级系统。

6.1 概述

随着中国经济的不断发展和国民素质的提升，居民收入水平和消费水平提高，预先获得物质享受的信用消费观逐步地被政府、商业部门、百姓所接受。实施良性的超前信用消费有利于拉动消费需求，促进国民经济发展；也有利于在全社会建立信用机制，鼓励国民依靠自身能力和素质去创造财富。促进良性超前信用消费的关键是建立多层次的信用评定和制约机制。

就金融领域而言，建立客户信用评定机制，不仅有利于为优质客户提供信用服务，并有效防范信用风险，更有利于开展新的客户服务品种，开拓市场，提高管理水平和竞争力。

信用评估是授信者利用各种评估方式，对受信者在信用关系中的履约趋势、偿债能力、信誉状况、可信程度进行公正分析、审查和评估的活动。信用记录主要包括在公司中的信用记录、银行及其它信用数据中的记录、公安等司法部门的记录等。

6.2 预测评价指标体系

要进行个人信用评级，首先要考虑的问题是：个人的哪些信息可以作为个人信用评级的指标？如何将这些指标进行分类及量化处理？

信用等级是个人信用状况的一种反映，由于评定机构的目的不同，在评定的方法、内容等方面就有所不同，但各评定机构对个人评级都要通过一些特定的指标来进行评价，如个人月收入、供养人口、职业、工作单位性质等。

个人信用综合评估的指标体系是对个人信用进行综合评价的依据和标准，是综合反映个人本身和环境所构成的复杂系统的不同属性的指标，按隶属关系、层次结构有序组成的集合。由此，可将影响个人信用等级的因素加以系统分析与合理综合，建立如图 6-1 所示的综合评价指标体系。

由图 6-1 可见，这里针对评估对象将评估项分成五大类：自然情况、职业情况、家庭情况、社

会情况、特别记录情况等。

自然情况指评估对象的年龄、婚姻状况、文化程度、供养人口、户籍以及住宅性质等个人基本情况。

职业情况指评估对象的职业及其稳定性、在现单位工作年限、职务、职称和收入等情况。

家庭情况指评估对象的配偶、子女、家庭资产等情况。

社会情况指评估对象的社会参保、个人所得税费用、通讯费用等情况。

特别记录情况指评估对象在公安、检察、法院等司法机关的司法记录。

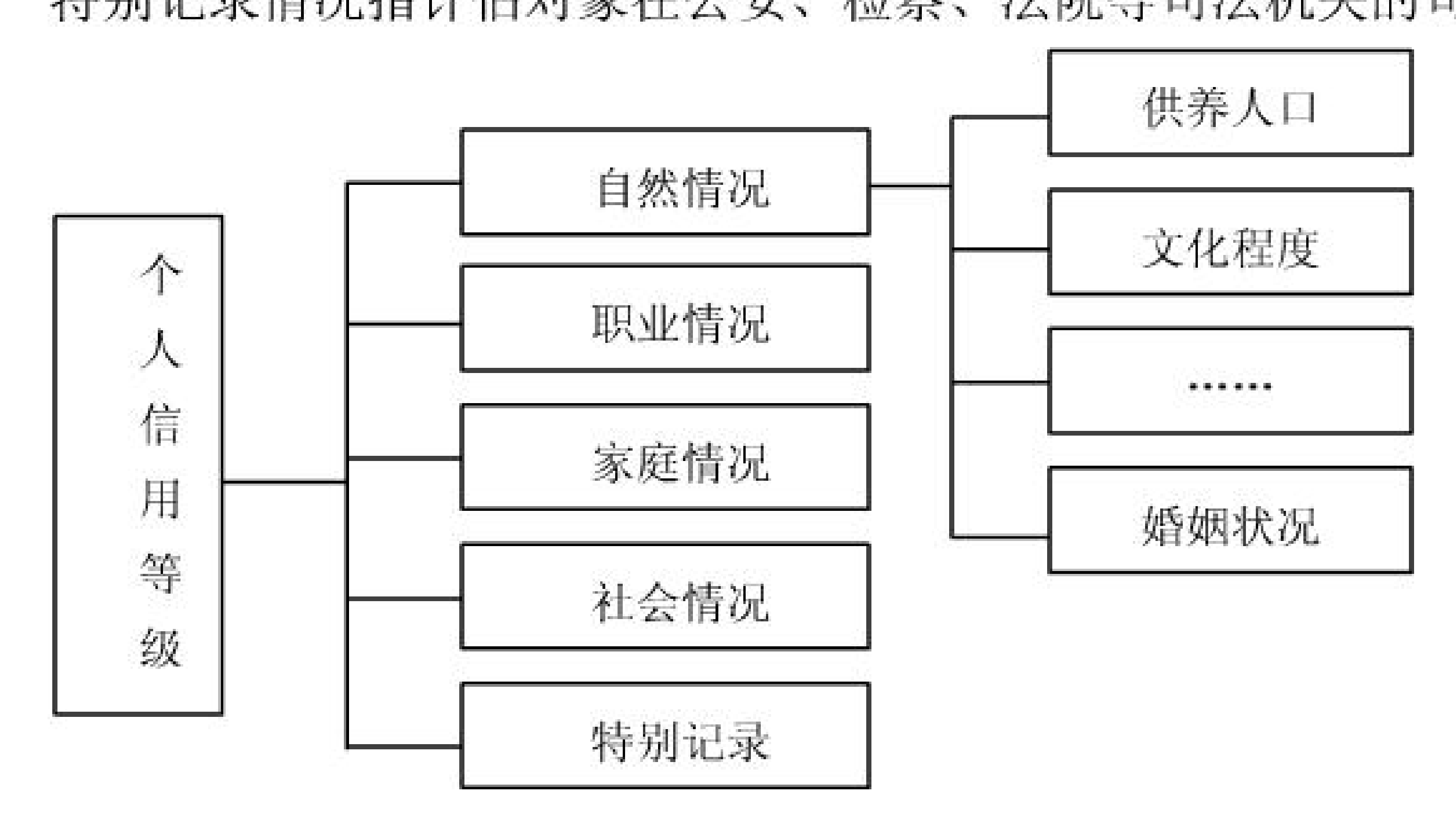


图 6-1 个人信用评估指标体系

6.3 预测评估模型

个人信用模型一般是建立在数据（信用记录）基础上的，是观察个人以往信用状况的记录，并依据个人经济实力、品德等能反映个人信用的特征因素，而建立的数理统计模型。目前，针对个人信用，各评定机构常用的方法是通过打分法，然后利用加权平均的方法得到客户信用评估得分。这种线性加权模型的个人信用评估方法的优点是使用简单，但是有很大的局限性。国内外大量的实证结果表明，个人的实际信用评估的各个因素之间有些相互影响，有些相互独立，呈现出复杂的非线性关系，用上述方法做出的评估比较粗糙。

资信评估的准确性如何，关键在于评估模型的建立，根据本案例的实际情况，我们有针对性地建立了如图 6-2 所示的评估模型：

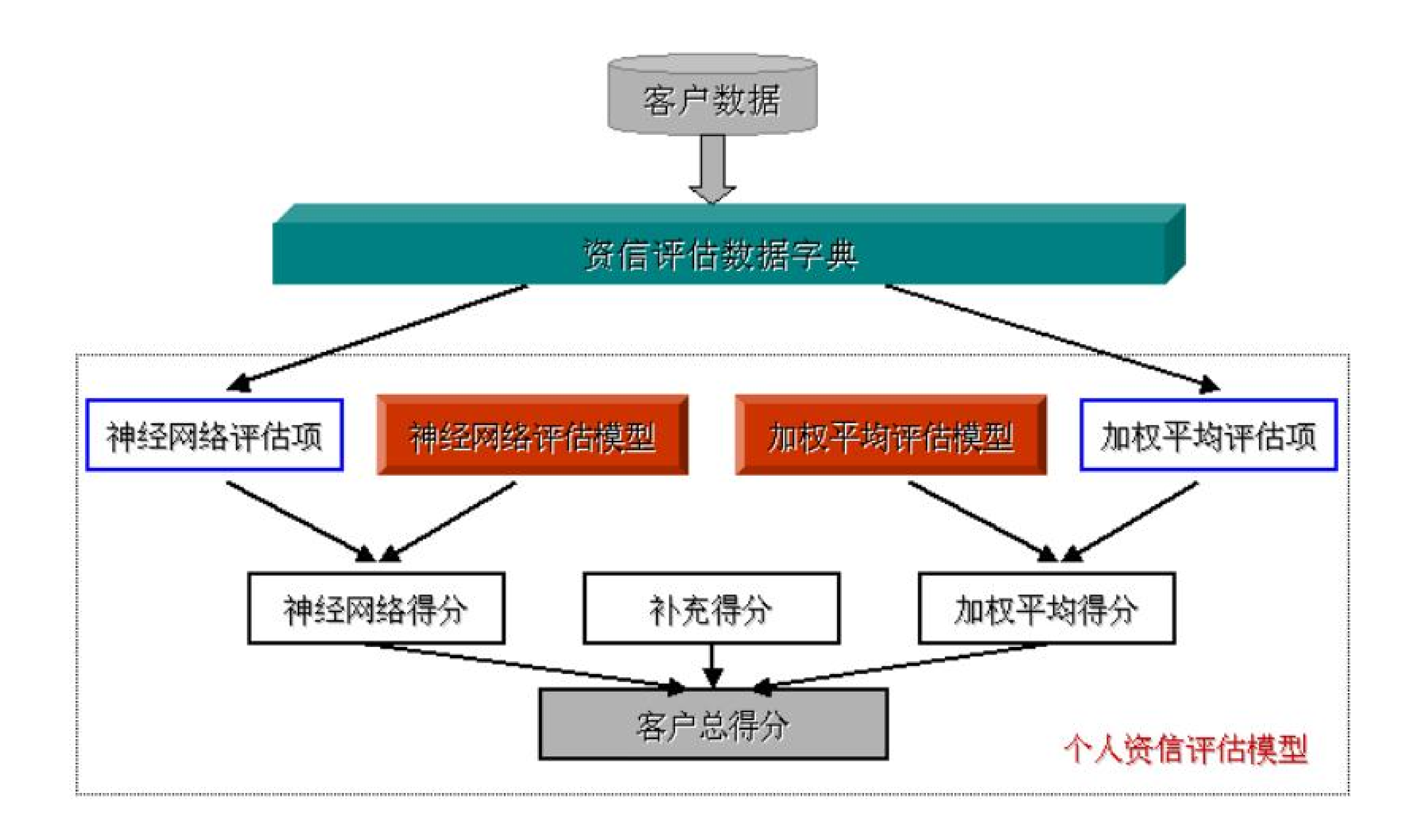


图 6-2 个人信用评估模型

由图可知，客户总得分由客观得分（神经网络得分和加权平均得分）和主观得分（由信控主管补充得到）组成。

神经网络得分通过神经网络评估项和神经网络评估模型得出，这里的神经网络评估项包括：客户年龄、年薪、催缴兑现率（客户透支后，通过催缴，客户如期归还次数占总透支次数的百分比）、上黑名单指数（在一定时期内客户上黑名单的次数）、上次评估得分等构成；加权平均得分通过加权评估项和加权平均评估模型得出，这里的加权平均评估项包括：户口类别、婚姻状况、配偶职务、配偶年收入、供养人口、住宅性质、汽车情况、单位类型、单位是否参加医疗保险、职位、职称等构成。下面章节主要介绍神经网络建模型。

说明：除了上面介绍的算法外，也可考虑其它算法，如多维时间序列方法：信用评估机构积累了大量单因变量 y（客户信用评估分数）、多自变量 xi（客户年龄、年薪、职务、婚姻状况等）的多维时间序列数据，这些数据既隐含大量的动态特征（需采用时间序列分析），又受环境因子的影响（需采用回归分析），同时具有高度的非线性（需采用非线性分析），这就要求建立一种基于结构风险最小、既反映样本集动态特征又体现环境因子影响的高精度非线性多维时间序列预测方法，即融合时间序列分析和回归分析的综合方法。如：基于相空间重构的数据挖掘时间序列；基于 SVR 的多维时间序列（SVR-CAR），本章将不对此方法做深入讨论，希望做更深入了解的读者可与作者联系。

6.4 有效模式和样本集的确定

模式是具有较强可比性和相似性的样本的集合，模式样本数量满足建立一个评估模型要求的模式称为有效模式，否则称为无效模式。

样本集是按照一定的规则划分的具有某种相同属性的样本集合，样本集在样本的某一属性上具有同一性。在个人信用评估中，如按地理区域的不同，可以划分出不同的样本集，按行业的不同也可以划分出不同的样本集。所以在实际操作中，有必要对个人数据进行分类，筛选出不同的样本集。

因为样本跨越不同的地理区域、行业，每个财务指标在数值上的差异非常大，如果不剔除掉那些偏离中心太远的样本点，整个样本集的数据势必会受到这些奇异样本的影响，使整个样本集的信息受到弱化。所以在选取样本时，必须考虑样本中奇异点的剔除。

在确定了个人信用评估指标体系后，我们从资信评估公司的个人信用信息系统中筛选出某地理区域的个人数据，并将样本中的奇异点剔除，组成一个符合有效模式条件的样本集，以此作为后续训练神经网络模型的专家样本数据，构建专家样本库的具体操作步骤如图 6-3 所示。

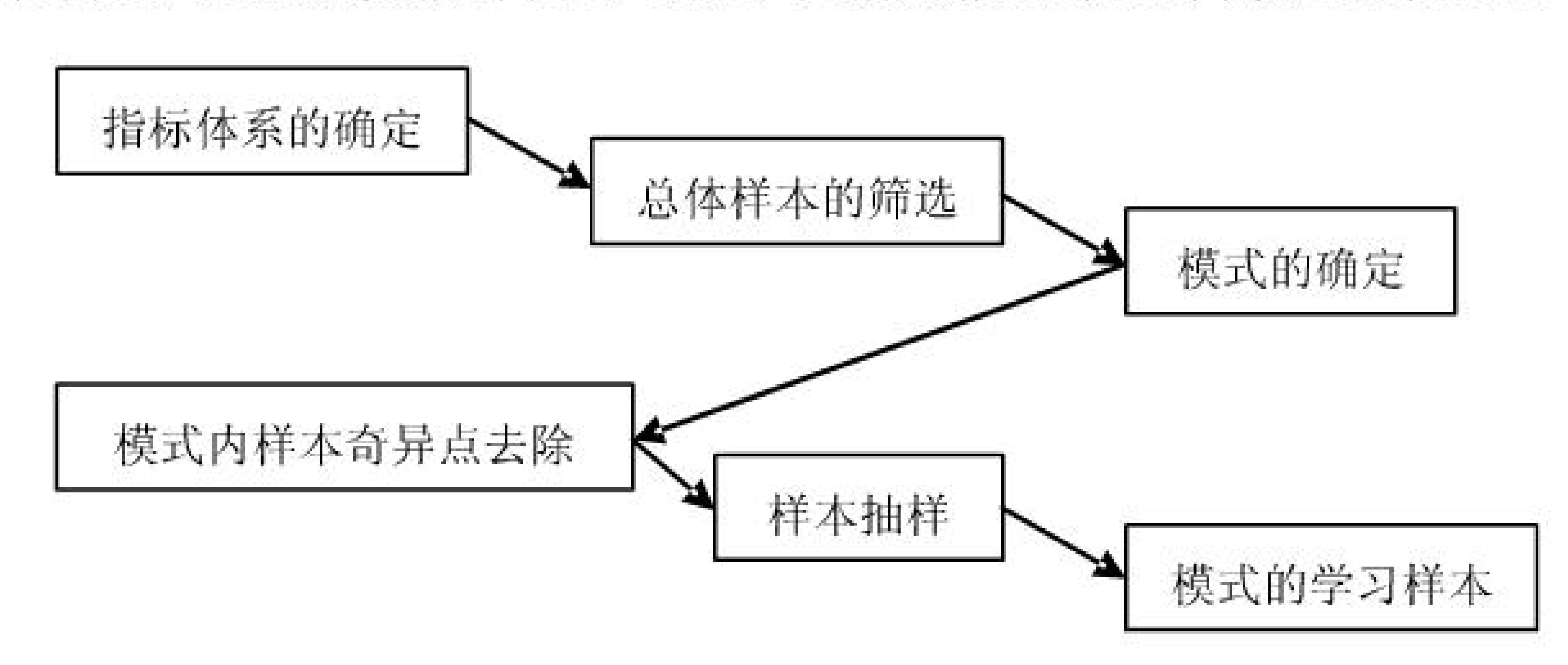


图 6-3 构建专家样本库流程

6.5 样本库的建立和归一化处理

样本库需要事先建立，每个样本由输入向量和输出向量构成。输入向量包括客户年龄、年薪、催缴兑现率、上黑名单指数、上次评估得分等。输出向量是客户信用得分。

6.5.1 样本库的建立

在实际应用中，往往由于没有历史样本数据或数据不足，即不具备建立有效模式的条件，使得神经网络算法很难在实际应用中得以推广应用。

如果存在大量的历史数据，我们必须对这些数据进行筛选和过滤，使之符合建立一个有效模式的样本集合。哪些数据应该保留，哪些数据应该过滤，说明模型的输入与输出之间本来就存在某些规则和关联，而这一操作过程本身就带有主观性。所以即使在没有历史样本数据的情况下，我们也可以在模型的输入与输出之间，定义某些关联规则，即通过主观产生建立一个有效模式的足够的样本数据，模型一旦在实际中得到应用，在日后的样本数据维护中，随着历史数据的增多，可逐步对专家样本数据进行补充和完善。

本例中，因为历史样本数据缺乏，故主要通过经验，事先自定义规则来产生样本数据。

6.5.2 归一化处理

归一化的具体作用是归纳统一样本的统计分布性。归一化在 0~1 之间是统计的概率分布，归一化在-1~+1 之间是统计的坐标分布。归一化有同一、统一和合一的意思。无论是为了建模还是为了计算，首先基本度量单位要同一。

由于 BP 神经网络的隐含层一般采用 Sigmoid 转换函数，为提高训练速度和灵敏性以及有效避开 Sigmoid 函数的饱和区，一般要求输入数据的值在 0~1 之间。因此，要对输入数据进行预处理，而且对不同变量要分别进行预处理。如果输出层节点也采用 Sigmoid 转换函数，输出变量也必须作相应的预处理。因此为了取得最佳的学习效果，在进行神经网络训练前，必须对输入样本和输出样本进行归一化处理。目标：使归一化后的输入值和输出值较均匀地落在(0,1)区间内，这就要求对于特定的样本数据必须采用特定的归一化函数来处理。比如对于年薪这一输入向量，如果采用线性归一化处理方法，就很难达到很好的归一化效果。因为年薪的分布范围，绝大部分分布在 1 万元至 10 万元区间，但也有少于 1 万元和大于 10 万元的，如 2000 元，100 万元，这些样本是少数，但在建立专家样本库时这些样本不能忽略，而是都需要考虑进来，因此对于这类输入变量就不能简单采用线性归一化来处理。表 6.1 是作者根据经验总结出来的一些常用的归一化方法。

表 6.1 常用的归一化方法

归一化方法	计算公式	说　明
线性函数归一化	y=(x-MinValue)/(MaxValue-MinValue)	x、y 分别为转换前、后的值，MaxValue、MinValue 分别为样本的最大值和最小值
对数函数归一化	y=log10(x)	以 10 为底的对数函数转换

反余切函数归一化	y=atan(x)*2/PI	反余切函数转换

说明：训练时，需要对样本进行归一化，但如果是仿真，则需要对仿真的结果进行反归一化处理。

6.6 系统实现

个人资信评估系统在结构功能上分为两个部分：预测分析管理子系统，预测模型建模仿真子系统，如图 6-4 所示。

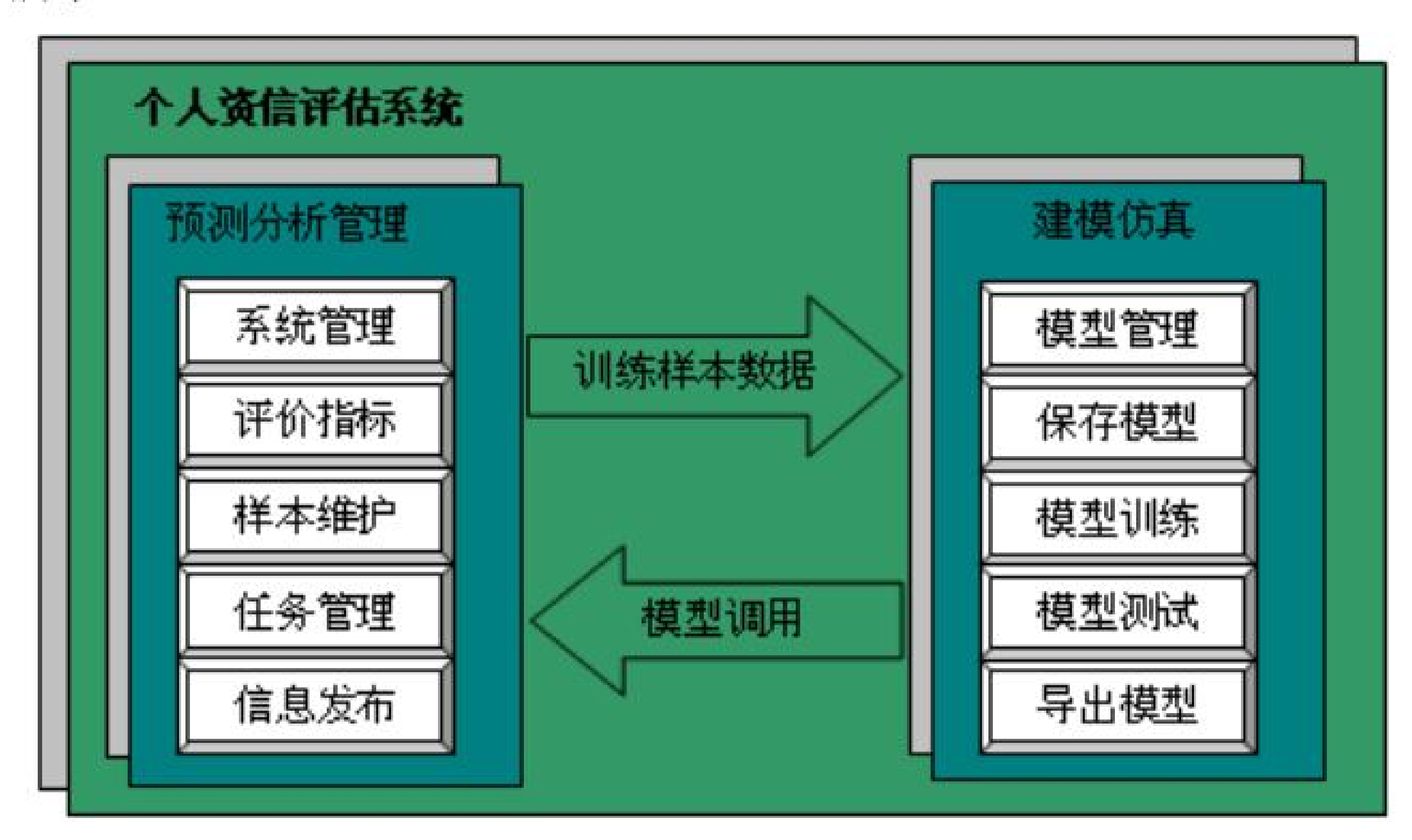


图 6-4 个人资信评估系统结构

预测评估管理子系统是个人资信评估系统的管理部分，负责维护预测分析的基础数据，如预测评价指标，专家样本数据等，并将专家数据传递给预测仿真子系统，同时调用来自预测仿真子系统已经建好的预测模型，并进行智能预测和分析。

预测模型建模仿真子系统是一个可视化的神经网络建模仿真工具，它基于 nnToolKit 神经网络工具包开发而成。主要负责神经网络模型的创建、训练及仿真测试，对于经过训练并经测试符合要求的预测模型可以导出，供外部应用程序调用。为方便读者学习和理解，前一章分别以 VC、VB、CB 等开发语言为例，详细介绍如何利用 nnToolKit 神经网络工具包经 MATLAB COM Builder 打包后生成的 COM 组件，来实现神经网络建模仿真系统。

练习题

1、常用的归一化方法有哪些？各适用什么场合，请举例说明。

2、在实际应用中，你将如何获取或组织专家样本数据？

3、什么是有效模式？什么是样本集？样本集如何确定？

附录 2NDN 神经网络建模仿真工具

前几章介绍了神经网络技术及其编程方法，神经网络的编程是不是只能基于 matlab 来进行？答案当然是否定的。下面介绍一款当前比较流行的神经网络建模软件——2NDN。

1 2NDN 神经网络建模型仿真工具简介

2NDN 神经网络建模仿真工具是一款可视化的神经网络建模应用开发平台，它基于行业标准设计和可连接组件的模块化结构实现，可扩展性强，可用于各种行业的运营管理和系统分析，是生产效率改善的有效工具，尤其适合高校建立系统仿真实验室。

2NDN 提供了 40 种以上的视觉化类神经组件，是一种可让使用者任意连接及合成不同的网络架构以实现类神经网络仿真及专业化应用，兼具视觉化美感的操作界面及强大功能的专业化软件。

它可以协助用户快速建构出所需的神经网络，并方便训练和测试所生成的网络模型。

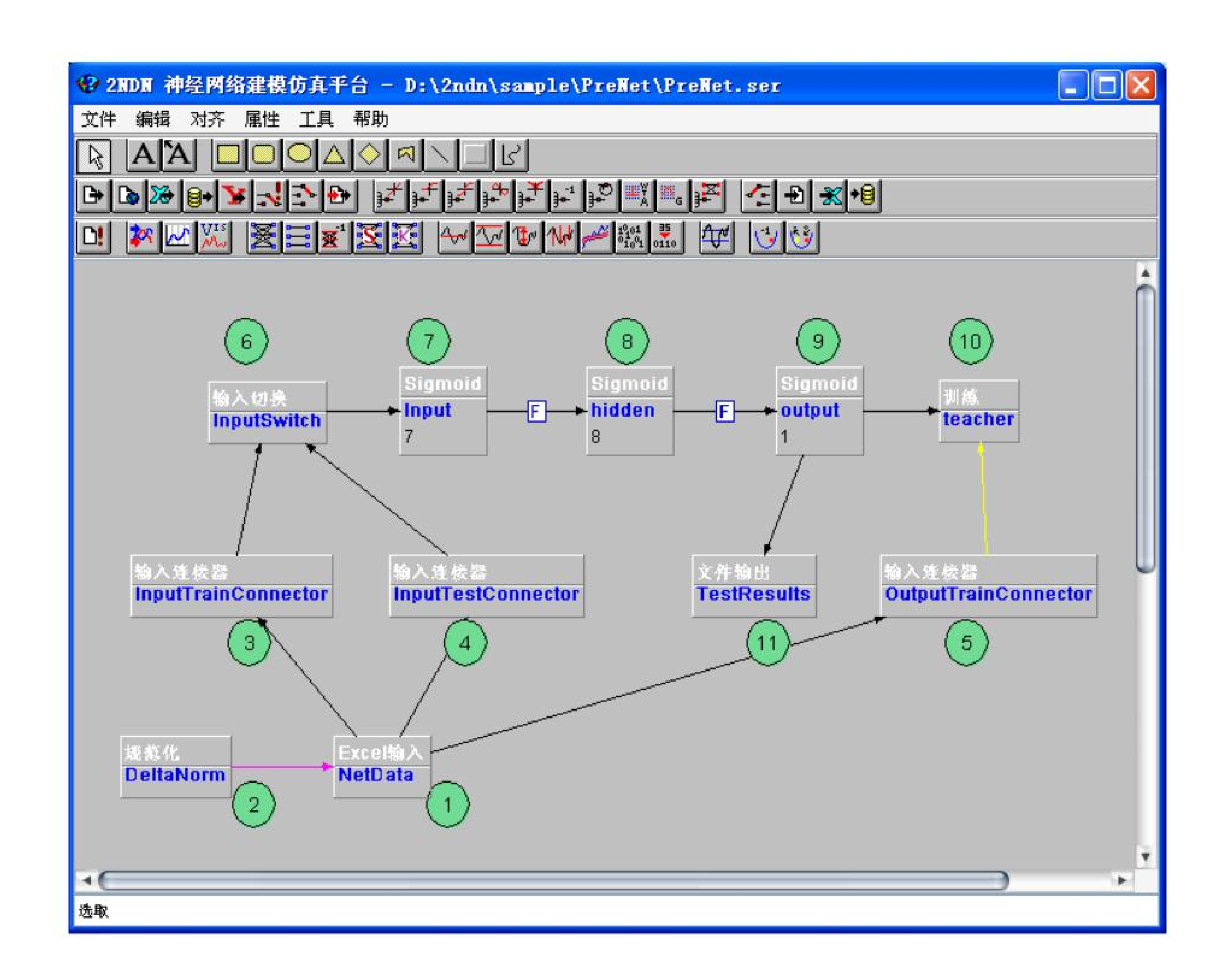
1.1 2NDN 主要特点

- **操作方便、易用** 可视化的人机交互界面，通过拖拉方式快速构建出任意复杂的神经网络模型，提供“克隆”技术，可有效地提高建模效率。
- **无缝集合和结果共享** 由于 2NDN 的数据来源和结果输出可以是文本文件、EXCEL 电子表格、URL 和 JDBC 数据源，所以它能与企业应用程序和其他统计工具无缝结合。快速读取源自第三方应用程序的数据。例如，批量评分，通过数据挖掘算法得出移动用户流失的数学模型，对每一用户进行流失可能性评分，系统把评分规则自动编码生成程序（SQL，XML 等），通过行业标准应用程序接口把模型移植到操作系统或者把评分回写到数据库，因而整个组织可以从数据挖掘过程的结果中受益。使用任意开发环境，数据挖掘可嵌入到内部和垂直的市场应用程序。
- **编程接口** 2NDN 提供一套基于 J2EE 行业标准的编程接口。它可用于开发各类数据挖掘应用程序，从简单的脚本到庞大的集成系统。数据挖掘引擎可由 JDBC 和 XML 访问分析行业标准数据挖掘 API。由 2NDN 建模工具创建的网络模型可保存为 SNET，被各种企业应用程序所使用。

- **基于插件的可扩展机制** 系统由组件构成，组件可组装，可重用，具有可伸缩性、可测量性，其内核引擎可被用户创建的自定义类所扩展和控制。这种可扩展性是通过使用一种能绑定的神经网络中一些组件上的插件来实现的。2NDN 提供三种主要的插件：输入插件；输出插件；监控插件。
- **跨平台的运行能力** 系统的运行平台包括 Windows、Unix 和 Mac 操作系统。

1.2 2NDN 功能简介

读者可以到 www.2nsoft.cn/2ndn 上下载最新版本的 2NDN，安装完毕后，运行时主界面如附图 1 所示。



附图 1 2NDN 运行时主界面

界面上菜单项排列在顶部。点击某个主菜单项，其子菜单项将展开，用户可以点击子菜单项来进行相应操作。工具栏放置在菜单项的下方，工具按钮由三组构成。第一组包含所有的绘图按钮，另外两组包含所有的神经网络结构组件。通过将这些组件连接到一起，可以建立任意复杂的神经网络模型。组件分为七类：输入层、输入切换层、神经网络层、输出转换层、输出层、训练层、连接层、图表层与插件。

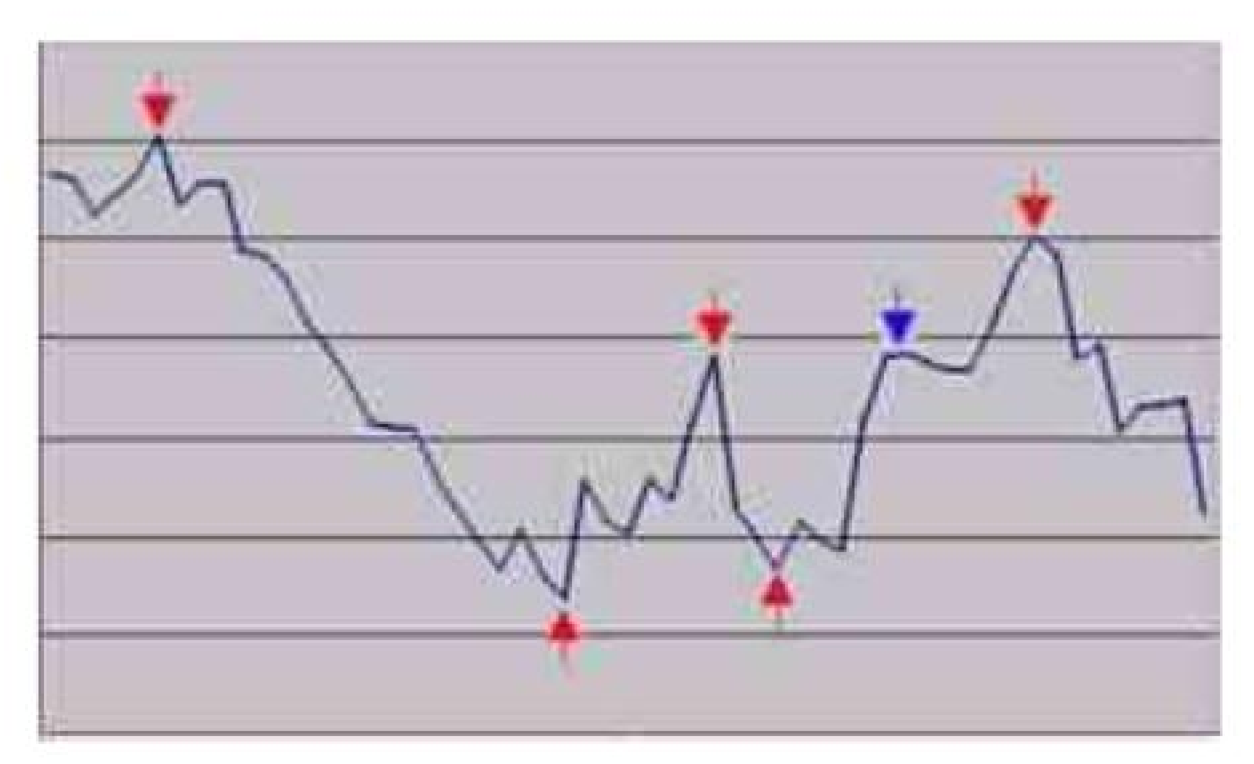
每个组件按钮的具体功能可以参考系统中的帮助选项，这里不做详细介绍。

工具条下面的灰色区域就是神经网络的建模区域，处理单元能够从工具条中拖到建模区并进行操作。下面通过一个案例介绍 2NDN 软件的使用。

2 基于时间序列的股票趋势预测模型

股票趋势预测代表经济预测中的一种特定类型，它具有其独有的特点。因为实际操作中不需要准确地知道下一天的收盘价，只需知道被观测市场的走向（升或降），从而决定交易操作（长期/短期－买入/卖出）。我们所希望的是利用历史价格数据预测将来中短期（从 2 到 10 或 15 天）内的价格走势。因而我们的神经网络也不需要预测下一个交易日的准确收盘价，从预测的趋势上去发现我们的交易策略。

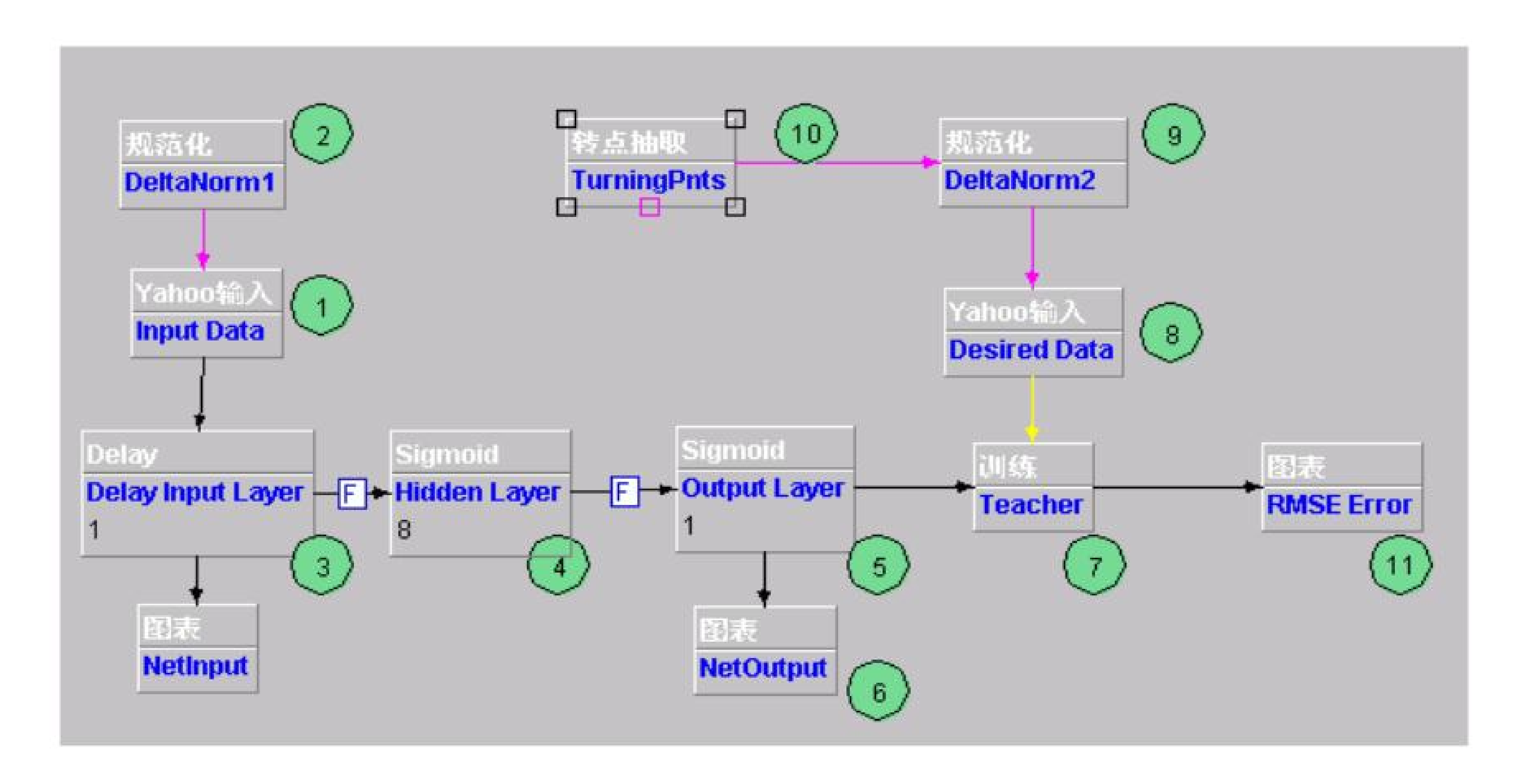
换句话说，需要预测的是市场的拐点。附图 2 是一只股票的日 K 曲线，要成功进行股市操作就要根据红色箭头来交易，在最低点买进，在最高点卖出。一个好的交易系统应当只是真正的拐点出现时给出提示，避免生成虚假信号，例如，象蓝色箭头所指部分，市场只下降了几个百分点就马上又继续上升。



附图 2 某股票的日 K 曲线

可见，所要设计的神经网络不需要预测市场的日准确收盘价，我们只对正确预测拐点感兴趣。下面就利用 2NDN 软件来实现基于时间序列的股票趋势预测建模。

打开 2NDN 神经网络设计工具，在主界面中通过拖拉方式添加下图所示的图元，这样神经网络模型就建好了，如附图 3 所示（为了便于学习，在各图元旁加了编号），同时 2NDN 会自动地随机设置连接的权系数。



附图 3 基于时间序列的股票趋势预测建模

下面对设计器中的各图元说明如下：

（1） Yahoo 输入 (Input Data)

用于神经网络从 yahoo 接收金融市场得来的金融数据，设置方法：鼠标选中该图元，右键弹出快捷菜单，点击<属性>，弹出如附图 4 所示的对话框。



附图 4 模型输入设置

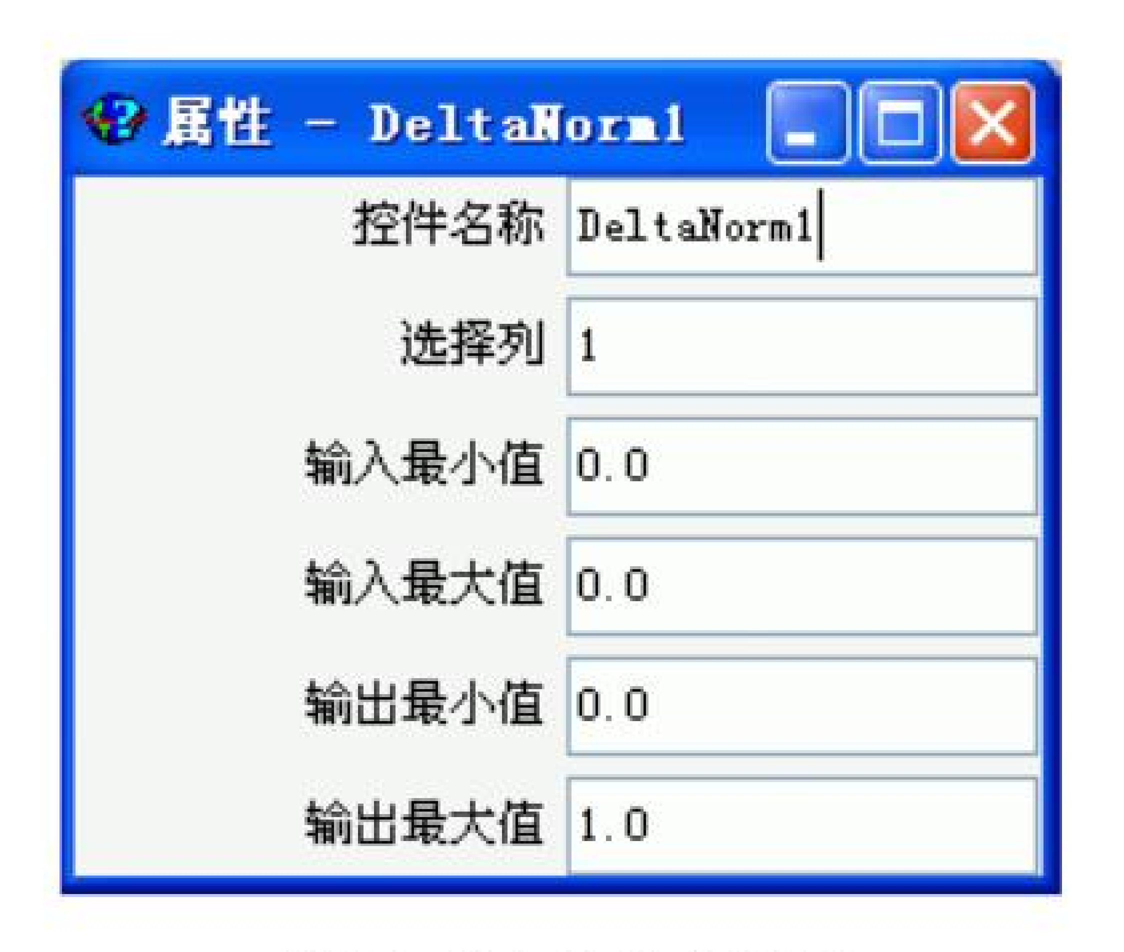
其中“符号”是指特定股票的代号。如 600728.SS 是指新太科技 600728 的股票。这个代号必须是由 Yahoo 定义的代号。“时期”指从 Yahoo 获取股票市值的周期，“Daily”获取每天收市记录的股票市值，“Monthly”获取每月月初记录的股票市值，“Yearly”获取每年年初记录的股票市值。“选择列”有六种可选项，分别表示 Open、High、Low、Close、Volume、Adjusted close。

对于特定的股票代码，用户必须根据所需的列来设置“选择列”。如果希望将 open，high 及 volume 列作为输入，则需将“1-2,5”写到“选择列”。

其它设置参考 2NDN 帮助文档说明。

（2） 规范化 (DeltaNorm1)

由于从 Yahoo 接受的数据不在神经网络能处理的范围内，因此必须先对输入样本数据进行归一化处理，设置方法：鼠标选中该图元，右键弹出快捷菜单，点击<属性>，弹出如附图 5 所示的对话框。



附图 5 输入规范化设置

说明：若输入最大/最小值设置为 0，表示自动取相应节点的最大/最小值。

附图 6 为经规范化处理后进行查验的结果。



附图 6 规范化后结果查验

（3） Delay (Delay Input Layer)

这里，使用 YahooFinance 输入组件从 Yahoo 获取股票价格的时间序列，将其与一个延时层相连。

设置方法：鼠标选中该图元，右键弹出快捷菜单，点击<属性>，弹出附图 7 所示的对话框。



附图 7 延时层设置

在这个组件的属性面板中，除“神经元数”之外，还可以设置“taps”参数，它是指我们用来提供给神经网络的时间窗口的大小。

如果设置 taps 为 5，就得到一个尺寸为 6 的时间窗口，它由下面的值组成：

[x(t), x(t-1), x(t-2), x(t-3), x(t-4), x(t-5)]

注意时间窗口的尺寸等于 taps+1，因为延时层还同时输出时间序列的当前值 x(t)。

（4） Sigmoid (Hidden Layer)

网络隐含层， 设置方法：鼠标选中该图元，右键弹出快捷菜单，点击<属性>，弹出附图 8 所示的对话框，设置其节点数为 8。



附图 8 隐含层设置

（5） Sigmoid (Output Layer)

网络输出层，设置方法参见（4）。

（6） 图表 (NetOutput)

这个组件用来在测试阶段显示出网络的预测结果曲线。设置方法：鼠标选中该图元，右键弹出快捷菜单，点击<属性>，弹出附图 9 所示的对话框。



附图 9 预测结果曲线设置

说明：X 轴最大值要大于网络所使用的金融数据的天数。

（7） 训练 (Teacher)

神经网络训练层，属性设置如附图 10。



附图 10 网络训练层设置

（8） Yahoo 输入 (Desired Data)

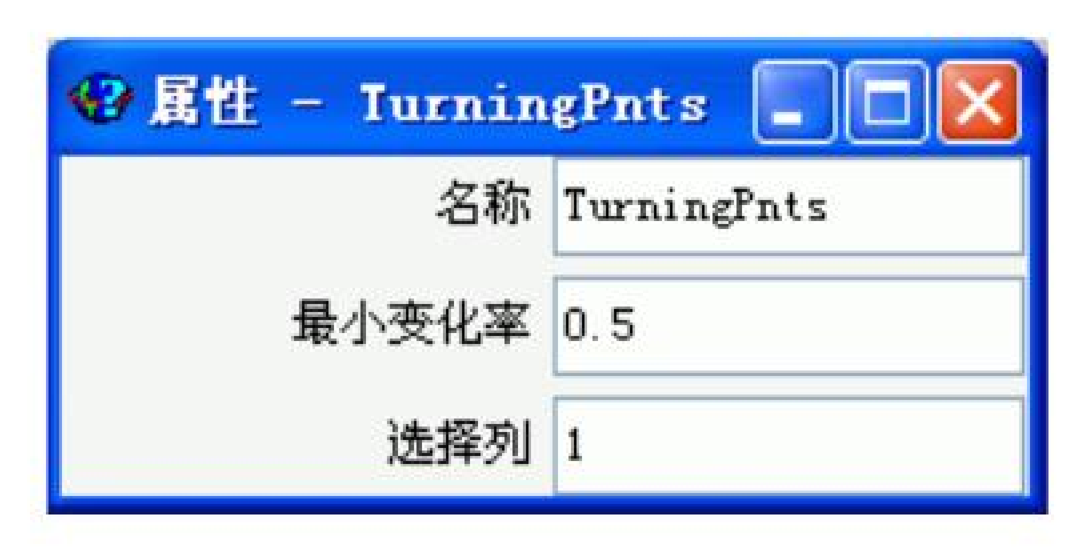
用于神经网络从 yahoo 接收金融市场得来的金融数据，用于生成训练网络的拐点信息。设置方法同（1）。

（9） 规范化 (DeltaNorm2)

在此对训练目标样本数据进行归一化处理，设置方法同（2）。

（10） 拐点抽取 (TurningPnts)

生成用来训练网络的拐点信息，属性设置如附图 11。



附图 11 拐点抽取设置

“最小变化率”表示两拐点之间的最小变化率，用来生成相应的信号。它不能设置太小值，否则会生成太多的信号（其中很多都是错误信号）。

算法如下：

- 当市场价格上升超过期望的变化率时，前面一个低点就被标注为“买”信号，相应的输出值设为 0。
- 当市场价格下降超过期望的变化率时，前面一个高点就被标注为“卖”信号，相应的输出值设为＋1。
- 上述两点之间的日期对应的期望值以插值到 0 与＋1 的方式进行规范化。

（11） 图表 (RMSE Error)

均方误差曲线，用来显示训练过程中的误差变化情况。设置方法：鼠标选中该图元，右键弹出快捷菜单，点击<属性>，弹出附图 12 所示的对话框。

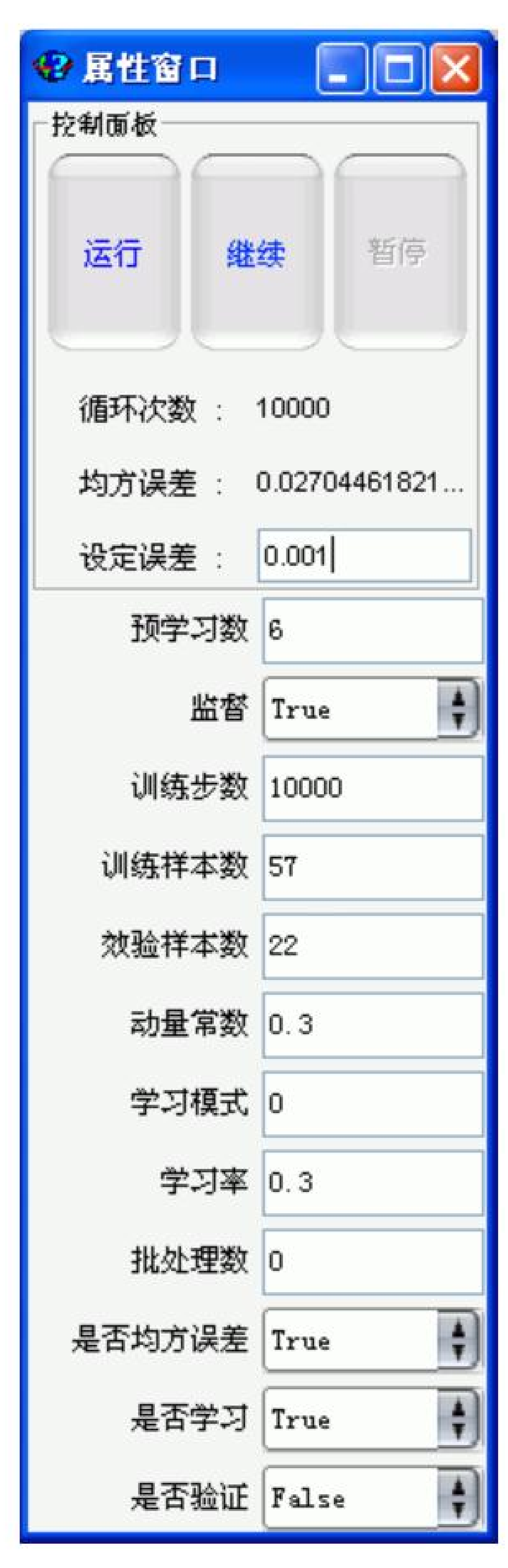


附图 12 误差曲线设置

说明：X 轴最大值要大于网络所使用的金融数据有天数。将“是否显示”设置为“True”时，将打开图表显示。

这里使用2007-6-1到2007-8-1的数据训练网络：将两个yahoo输入插件的时间段都设为2007-6-1至 2007-8-1。打开控制面板，设置学习率为 0.3，动量常数为 0.3，训练样本数为 41（该值可通过查验样本得到），训练步数为 1000 次，均方误差为 0.001，是否训练设置为 True，是否验证设置为 False，是否指导设置为 True，预测学习数设置为 6（该值为 taps+1），是否均方误差设置为 True。点击【运行】开始训练。控制面板上会显示训练的进度和收敛情况，见附图 13。如果权值参数选择不合理的话有可能造成网络不能收敛。这时可以选择【工具】—>【初始化】，重新随机生成初始的权系数。

说明：学习率是对神经网络训练至关重要的参数，太低会导致训练时间过长，太高可能会引起训练误差摆动，难以到达最优，选择正确的学习率意味着能得到好的训练结果，特别是对于时序预测模型。实际应用中可以利用动态退火插件来解决这一问题，在此不再介绍，读者可参考帮助文档对模块进行优化。

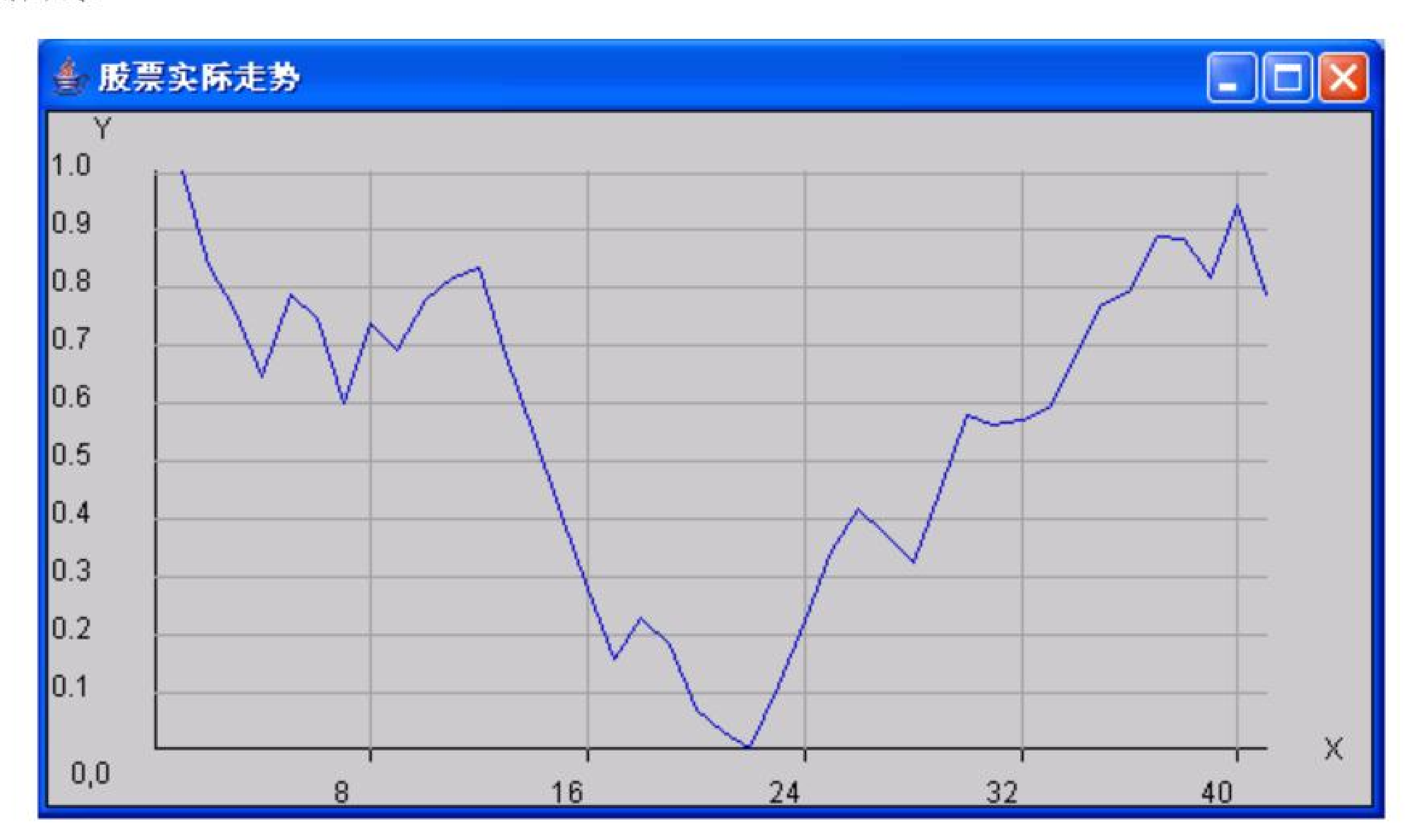


附图 13 控制面板

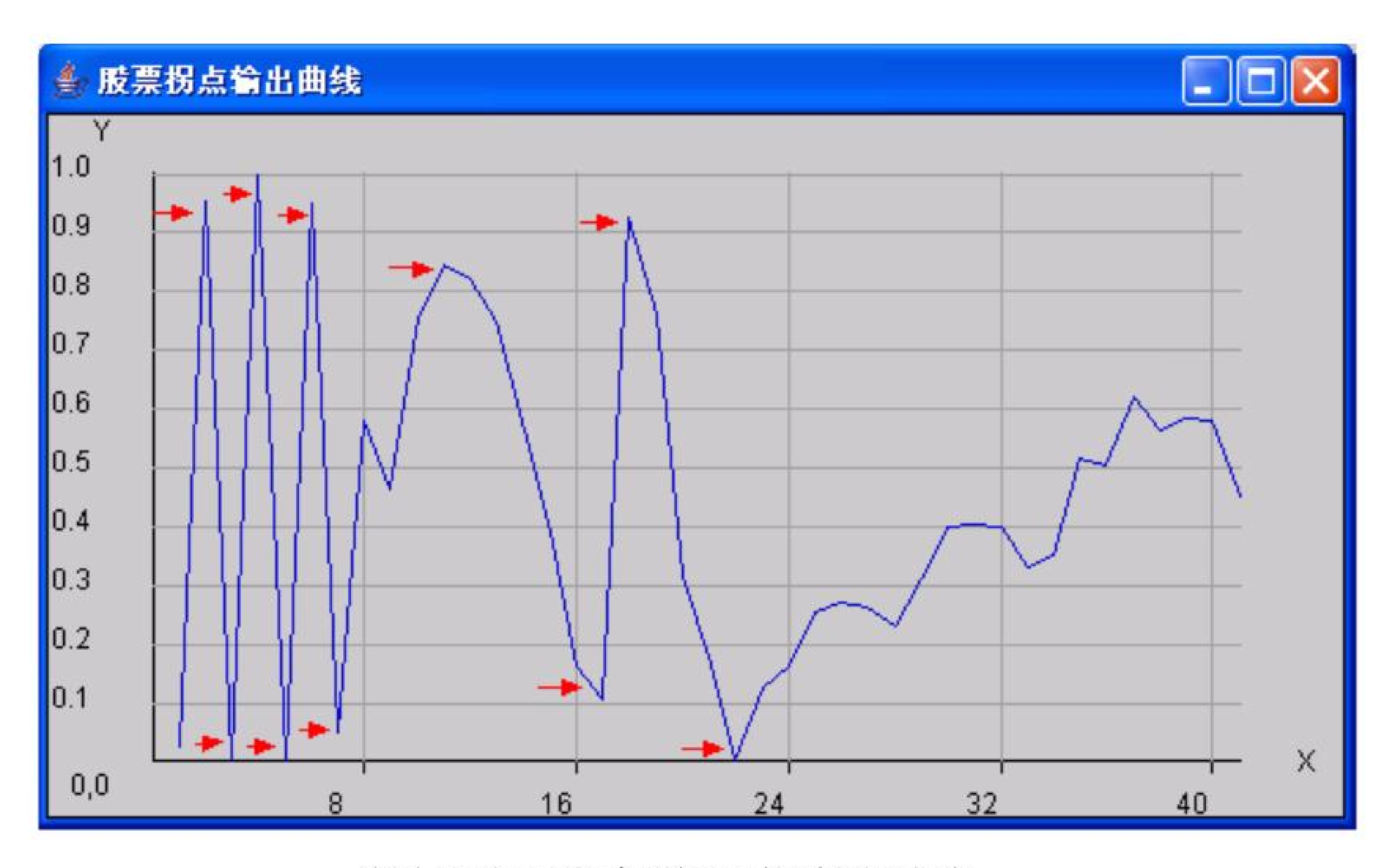
网络训练完成后，就可以对生成的网络模型进行测试了，我们使用 2007-7-1 到 2007-9-1 的数据作为测试数据，对该时间段内的股票价格拐点进行预测，测试时，将两个 yahoo 输入插件的时间段

都设为 2007-7-1 到 2007-9-1。打开控制面板，设置测试样本数（这里为 41），训练步数为 1 次，是否训练设置为 False，是否验证设置为 True。点击【运行】开始测试，预测结果将自动在图表插件中显示出来。

附图 14 和附图 15 分是 2007-7-1 到 2007-9-1 的股票实际价格走势和基于时间序列的股票拐点预测曲线。



附图 14 股票实际价格走势



附图 15 基于时间序列的股票拐点预测曲线

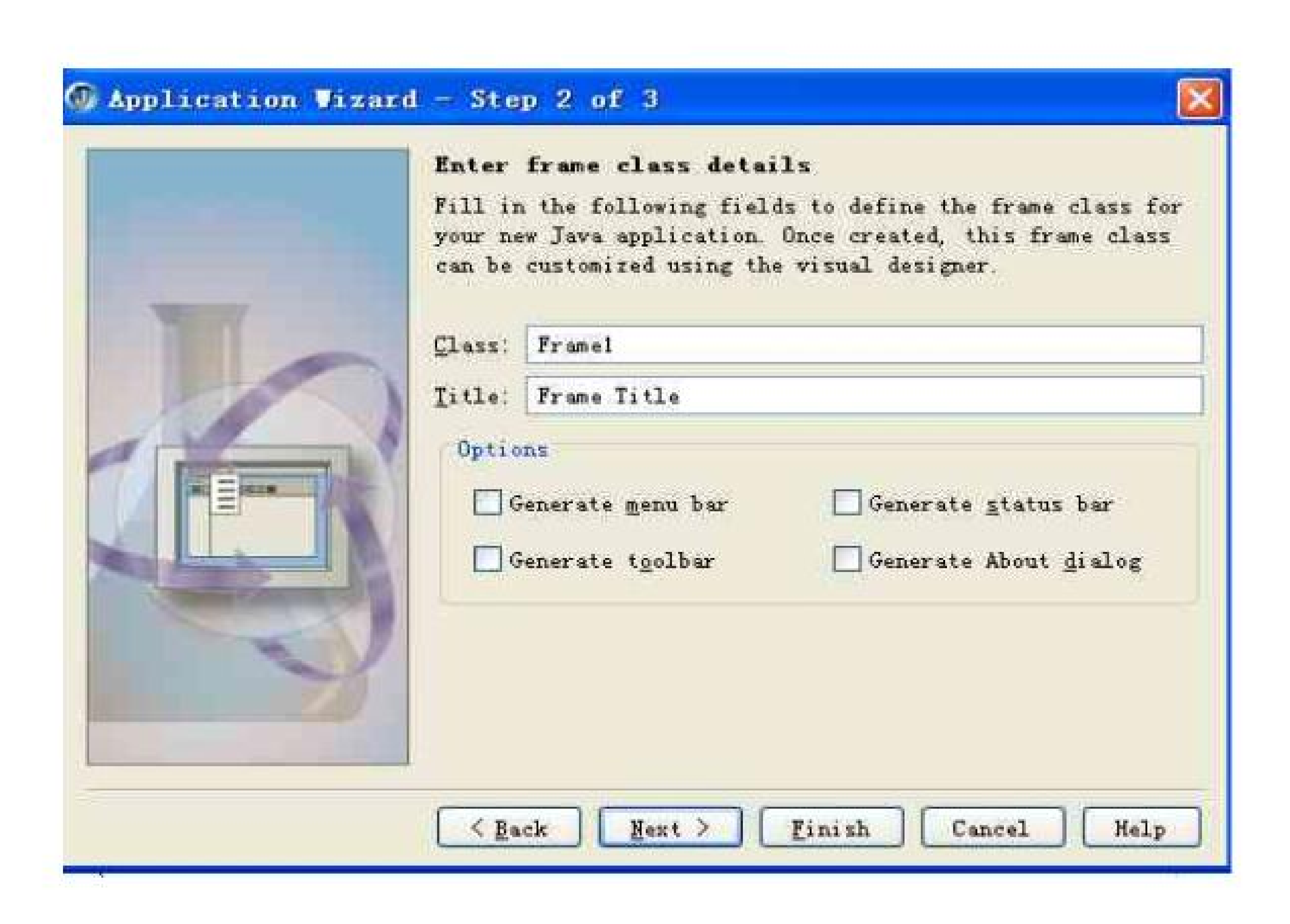
可以看出，该网络模型能够将绝大部分的股票走势的拐点正确预测出来，效果非常理想。图中箭头所标出的部分，预测转折指标量大于 0.8 或小于 0.2 的点经实际价格走势验证均为有效拐点。

3 用 2NDN 神经网络建模型仿真工具实现混合编程

前面两节分别介绍了 2NDN 的功能及案例，下面将介绍如何在程序中嵌入建好的神经网络模型。本节中，使用 Borland 的 java 开发工具 JBuilder2006 来开发一个小的桌面程序，通过嵌入 2NDN 建好的 XOR 神经网络模型，来实现异或运算的功能。关于 JBuilder2006 的操作，这里不做介绍，有兴趣的读者可以阅读相关方面的书籍。

1、建立神经网络模型 利用 2NDN 建立 XOR 神经网络模型并训练至较小的均方误差，导出为 xor.snet 文件。

2、开发用户界面 打开 JBuilder2006，新建工程 xorjb2006，在工程中新建应用程序 xorapp.java，设置 class 属性为“Frame1”。如附图 16 所示。



附图 16 创建工程应用

在工程 xorjb2006 中打开 Frame1.java 文件，单击界面下方的“Design”按钮，通过拖拉方式设计界面，其中包括两个 JTextfield 组件，两个 JLable 组件和一个 JButton 组件。按附图 17 所示调整组件位置、设置组件属性。



附图 17 窗体设计

设计好整个界面的布局，单击“提交”按钮，设置鼠标单击的属性，具体代码如下。

```
public void jButton1_mouseClicked(MouseEvent e) {
    try {
        //filename 表示模型 xor.snet 的存放路径，读者需要设置为本地的路径。
```

```
            String filename = "F:/developer/xor.snet";
            //从两个文本框中读取数据，并转换为 double 型变量。
            String x = jTextField1.getText();
            String y = jTextField2.getText();
            double doub1 = Double.valueOf(x).doubleValue();
            double doub2 = Double.valueOf(y).doubleValue();
            //设置数组 inputArray，用来保存从文本框中读取的数据。
            inputArray = new double[][] {
                    { doub1, doub2}
                };
            // eXOR 为类 EmbeddedXOR 的一个实例。
            eXOR.Go(filename);
           //设置提示框的属性，用以显示模型的运算结果。
           JOptionPane.showMessageDialog(this, "xor 神经网络模型运算结果：" + eXOR.array[0]);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "请输入数字 1 或 0");
        }
    }
```

3、嵌入 XOR 模型并编写功能代码 在工程 xorjb2006 中新建类 EmbeddedXOR.java，用来实现导入神经网络模型 xor.snet 并运算的功能，下面是这一部分的代码。

```
    public class EmbeddedXOR {
        Frame1 fr1 = null;
        public double[] array = null;
        //创建 EmbeddedXOR 的构造函数。
        public EmbeddedXOR() {
        }
        // restoreNeuralNet()方法用来读取建好的神经网络模型。
        private NeuralNet restoreNeuralNet(String fileName) {
```

```
            NeuralNet nnet = null;
            try {
                  FileInputStream stream = new FileInputStream(fileName);
                  ObjectInputStream inp = new ObjectInputStream(stream);
                  nnet = (NeuralNet)inp.readObject();
            }
            catch (Exception excp) {
                  excp.printStackTrace();
        }
        return nnet;
}

    // Go()方法利用导入的神经网络模型进行运算。
    public void Go(String fileName) {
        //导入模型，并设置为 NeuralNet 型变量。
        NeuralNet xor = restoreNeuralNet(fileName);
        if (xor != null) {
                  //使用 getInputLayer()方法得到模型的输入层 input。
                  Layer input = xor.getInputLayer();
                  //去除连接在 input 层上的输入层，例如文件输入层等。
                  input.removeAllInputs();
                  // memInp 用来为输入层 input 提供数据。
                  MemoryInputSynapse memInp = new MemoryInputSynapse();
                  //读取数组的的第一行数据。
                  memInp.setFirstRow(1);
                  //读取数组的 1，2 列数据。
                  memInp.setAdvancedColumnSelector("1,2");
                  //在输入层 input 上连接数据输入层 memInp。
```

```
            input.addInputSynapse(memInp);
            //设置数据源为 Frame1 中的数组 inputArray。
            memInp.setInputArray(fr1.inputArray);
            //使用 getOutputLayer()方法得到模型的输入层 output。
            Layer output = xor.getOutputLayer();
            //去除连接在 output 层上的输入层，例如文件输出层等。
            output.removeAllOutputs();
            // memInp 用来从输出层 output 获取运算结果。
            MemoryOutputSynapse memOut = new MemoryOutputSynapse();
            //在输出层 output 上连接数据输入层 memOut。
            output.addOutputSynapse(memOut);
            //设置模型的运算次数，本例中设为 1。
            xor.getMonitor().setTotCicles(1);
            //设置导入模型的运算样本，本例中设为 1。
            xor.getMonitor().setTrainingPatterns(1);
            //将模型学习的参数设为 false，本例中模型已经经过训练，不需要进行学习。
            xor.getMonitor().setLearning(false);
            //开始运算。
            xor.start();
            xor.getMonitor().Go();
            //将运算结果传递到 array 数组。
            double[] pattern = memOut.getNextPattern();
            array = pattern;
        }
        //停止运算
        xor.stop();
    }
}
```

4、程序的测试：经过上面的几个步骤，这个桌面程序已经完成。下面测试一下它的功能。运行程序，在文本框中输入 1，0 两个数据，如附图 18 所示。



附图 18 程序测试

单击提交按钮，弹出对话框，运算结果为 0.99294065，如附图 19 所示。



附图 19 程序运行结果

有兴趣的读者可以利用这里讲述的方法，通过 2NDN 与其它 javaIDE 工具的混合编程，结合本附录第二节讲述案例中的神经网络模型，开发基于具体应用的神经网络应用软件

练习题

1、用 2NDN 实现 4.5.1 节所示的房地产开发风险预测。

2、用时间序列算法实现 4.5.2 节所示的地基沉降量预测。

3、用 2NDN 实现 XOR (eXclusive-OR) 问题。

说明：XOR 问题实质是：如果两个输入项中的一个为 true，其输出为 true，但如果两个输入项

均为 false 或 true，则输出为 false。

其关系如图：

Input 1	Input 2	Output
false	false	false
false	true	true
true	false	true
true	true	false

参考文献

1 李弼强,彭天强,彭波等编著 智能图像处理技术 北京:电子工业出版社.2004

2 飞思科技产品研发中心编著 MATLAB 6.5 辅助图像处理 北京:电子工业出版社.2003

3 施阳,李俊,王惠刚,严卫生编著 MATLAB 语言工具箱--TOOLBOX 实用指南.1998

4 张兆礼,赵春晖,梅晓丹编著 现代图像处理技术及 Matlab 实现 北京:人民邮电出版社.2001

5 闻新,周露,李翔,张宝伟 编著 MATLAB 神经网络仿真与应用 北京:科学出版社.2003

6 罗四维编著. 人工神经网络建造 北京:中国铁道出版社.1998

7 韩力群编著 人工神经网络理论 设计与应用 北京:化学工业出版社.2002

8 杨铭震 王燕霞编著 人工神经网络及其在石油勘探中的应用 北京:兵器工业出版社.1993

9 飞思科技产品研发中心编著 神经网络理论与 MATLAB7 实现 北京：电子工业出版社. 2005